6-2019

MULTI-LEVEL SECURE DATA DISSEMINATION

Garo Panossian

MULTI-LEVEL SECURE DATA DISSEMINATION

A Project

Presented to the

Faculty of

JHB College of Business and Public Administration,

California State University

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Information Systems and Technology

---

by

Garo Panossian

June 2019

MULTI-LEVEL SECURE DATA DISSEMINATION

A Thesis

Presented to the

Faculty of

JHB College of Business and Public Administration,

California State University

San Bernardino

---

by

Garo Panossian

June 2019

Approved by:

Joon Son, PhD., Chair	Date
Essia Hamouda, PhD., Co-Chair,	Date
Conrad Shayo, PhD., Reader	
Jay Varzandeh, PhD., Chair, Information & Decision Sciences Department	

ABSTRACT

Multi-level security is prevalent within the military; however, the private sector has not yet invested in the approach. As big data, Internet of things, and artificial intelligence drive businesses to collaborate (share data, algorithms, and tools) the need to secure such resources while simultaneously sharing them will push towards an alternative approach-–namely Multi-level security. The military labels data according to the sensitivity it carries as related to national security. Furthermore, the military restricts access by both the overall trust in the individual and by their need-to-know. To put it another way, data has a certain level of sensitivity and only those individuals that can be trusted with the data and have a need-to-know shall have access to such data. Military organizations not only limit access to digital data but also to sensitive discussions, often having sensitive talks within a Sensitive Compartmented Information Facility referred to as a SCIF. Irrespective of the media, all data must be secured and disseminated in order to produce value. Inaccessible data has no real value, as data must be accessible in order to be actionable and produce value. Along the same lines, data often requires aggregation to become actionable.

Creating a security domain with multiple levels of trust and need-to-know ensures that data can both be accessed and aggregated. Multi-level secure domains exist in military organizations today, however, the challenge arises when two domains want to share data—hence the need for multi-level secure data dissemination. One way to accomplish this objective is for Domain X to contact Domain Y and together identify how their two security domains can map to one another. After determining the mapping Domain X can send Domain Y data, however, what if Domain Z wants access to the same data? Should Domain Z request the data from Domain Y? Would Domain Y violate

the trust of Domain X, if Domain Y disseminates the data? Perhaps, Domain Z is only cleared to a portion of the data. These are the issues related to the dissemination of MLS data within a multi-domain environment.

The objective of this project is to propose a solution that would allow domains to securely disseminate data without the need to repackage the data for each domain. The solution outlined in this project, leverages Simple Public Key certificates, Active Bundle, and a directory server. When combined, the three technologies allow domains: to convey both trust and authorization policies, learn about trust and authorization policies of external domains, and provide a mechanism to securely disseminate data.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Motivation

The issues of both securely storing and securely processing of sensitive data are paramount in a variety of sectors. Recent events highlight several security related issues. For example:

- Vatileaks (Wooden, 2012)
- Marriott Data Breach (Sanger, Perlroth, Thrush, and Rappeport, 2018)
- 2016 Presidential Campaign Hacking ("2016 Presidential Campaign Hacking Fast Facts", 2019)
- Snowden's disclosures

Moreover, technological progress along with the need to process, store, and share an ever-increasing amount of data brings about distributed systems capable of processing data from smart sensors and networks. We use applications and devices that constantly track and collect our personal information, these systems should be properly protected, and access limited to individuals with valid authorizations (Liguori, 2016). To achieve these objectives, many organizations, such as the military services, intelligence organizations, related government agencies, and their supporting defense industries require Multi-Level Security (MLS). These systems enable the concurrent processing of data at multiple and unique classifications. A security policy in a MLS environment must enforce a model of access control, known as the Bell-La Padula (BLP) model, which prevents individuals from accessing information at a classification for which they are

not authorized. Examples of such sensitivity or security levels could be Unclassified, Confidential, Secret, and Top-Secret.

Furthermore, the velocity and volume of available data are increasing through the contributions of both humans and devices alike. Also the demand for big data analysis are increasing, as businesses turn to big data analytics to create the competitive advantages needed within the marketplace. The transformation of data into information, then information into business intelligence costs time, money, and expertise. The end result of this transformation is the most valuable resource for any business–knowledge. However, the marketplace demands innovation and constant motion, stagnation causes atrophy. As businesses share their data with partners, loose employees to competitors, and invest heavily in new technologies, the demand for securing their data will grow. MLS is not common within the private sector now, as many businesses rely on the Windows operating system, which is based on discretionary access control (DAC). With DAC subjects can create files, thus becoming the owner, and pass on permissions to any subject of their choosing. However, as security becomes a greater concern for businesses the need to secure data at an organizational level will grow, leading to MLS. The growing need to keep some data or information 'confidential' for competitive advantage, trade secrets, abide by regulations, intellectual property, or commercial-in-confidence enforcement, is a strong driving force pushing towards a security-centric architecture.

Numerous studies have shown that unauthorized access poses a major security threat for distributed enterprise environments. The threat is exacerbated in a multi-domain environment where businesses collaborate in order to meet their business objectives. The challenge is in developing new or extending existing security models for efficient security management and administration in multi-domain environments that allow secure interoperation among individuals or organizations belonging to different security domains. More specifically, operating within a multi-domain environment, where each

domain has its own MLS policy, brings up two primary questions. First, what are the effects on security when two domains have mapped their MLS policies incorrectly? Second, how can both the sensitive data and the MLS policies be distributed amongst multiple domains securely?

Organization

The remainder of the project is organized as follows. Chapter 2 summarizes the literature review. Chapter 3 identifies several BLP security violations that arise when two domains map their MLS policies incorrectly. Since the literature review did not uncover the particular BLP violations related to incorrect mapping, six test scenarios are developed to investigate the violations. Chapter 4 outlines how MLS policies are represented within SPKI name and authorization certificates, along with demonstrating a Prolog implementation of the logic. Chapter 5 presents the MLS Cross-Domain Security Policies. Chapter 6 presents a solution that allows for the secure dissemination of MLS data and concludes with a summary including contributions, findings, and shortcomings.

CHAPTER 2

BACKGROUND

Mandatory Access Control

Mandatory access control (MAC) is a method of limiting access to system resources based on the sensitivity of the information (classification) and the authorization of the user (clearance). In a seminal work Denning, 1976; Bell, 2005 by Denning, a lattice structure was introduced to formally specify a Bella Padular (BLP) model (BellDavid, 2005) for a MLS system. The underlying goal is for information to flow from lower classifications to higher classifications (Example 1), while also providing the additional level of protection known as compartmentalization or "need to know" (Example 2). Below, in Figure 2.1 is an example of the lattice structure. The security levels are connected by edges, and information should flow from a lower classification to a higher classification following the edges.

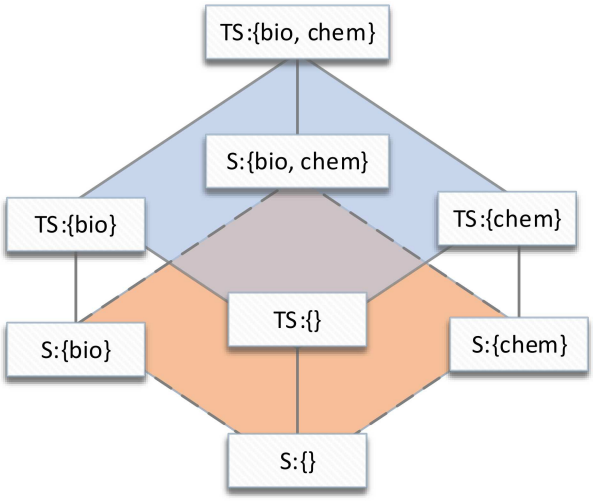


Figure 2.1: Lattice structure.

Example 1. *Based on Figure 2.1, $TS:\{\}$ is a higher classification than $S:\{\}$. Therefore, $TS:\{\}$ can read from $S:\{\}$ and $S:\{\}$ can write to $TS:\{\}$, allowing information to flow from a lower classification to a higher classification. However, $TS:\{\}$ cannot write to $S:\{\}$ nor can $S:\{\}$ read from $TS:\{\}$, as this would allow information to flow from a higher classification to a lower classification. The implementation mechanics are discussed in Examples 3 and 4.*

Example 2. *Based on Figure 2.1, $TS:\{\}$ cannot read from or write to $S:\{bio\}$. Although the Top Secret classification ($TS:\{\}$) is higher than the Secret classification ($S:\{bio\}$), $TS:\{\}$ does not have the need to know as it lacks the $\{bio\}$ compartment. By creating compartments, such as $\{bio\}$, access becomes much more granular, simply having a Top Secret clearance will not grant a user access to everything.*

To prevent information from leaking, the BLP models proposed: the simple security property and the star property. Taken together the two properties ensure information flows from low to high (Figure 2.2). The security policies are built around subjects (S) and objects (O), each with a security level. A subject's security level is referred to as a clearance, whereas an object's security level is referred to a classification. A subject can access (read, write, or modify) an object given as a set of security rules.

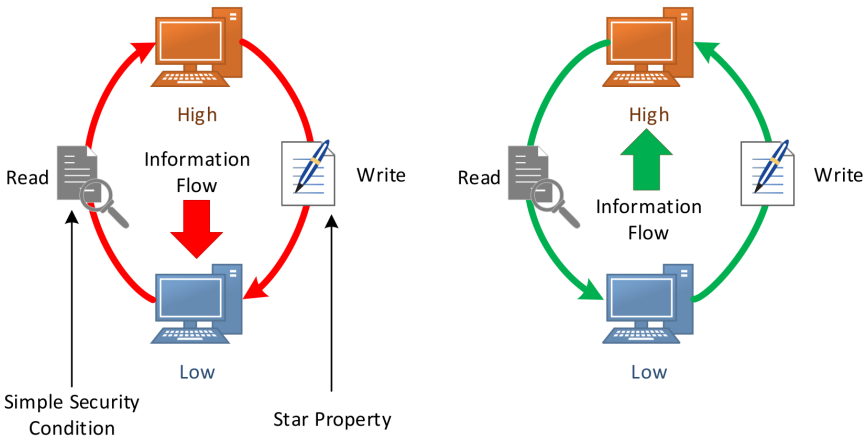


Figure 2.2: BLP properties and information flow.

Simple security property Also known as the "no read-up," states that a subject at a given security level may not read an object at a higher security level (Bell, 2005). In other words, a subject (S) may read object (O) if the security level (sl) of the object is less than or equal to the level of the subject.

$$sl(O) \leq sl(S)$$

Example 3. *Assuming Alice has the security clearance* $TS : \{bio\}$*, Alice cannot read* O_1 *(with the classification* $TS : \{bio, chem\}$*), as the security classification for* O_1 *is higher than Alice's clearance. However, Alice can read the object* O_2 *(with the classification* $S : \{bio\}$*), as* $sl(O_2) \leq sl(Alice)$*. Furthermore, Alice could also read* O_3 *(with the classification* $S : \{\}$*).*

$*$ **(star) property** Also known as the "no write-down," states that a subject at a given security level may not write to any object at a lower security level. In other words, the subject (S) may write to the object (O) if the security level (sl) of the subject is less than or equal to the level of the object.

$$sl(S) \leq sl(O)$$

Example 4. *Assuming Alice has the security clearance* $TS : \{bio\}$*, Alice can write to* O_1 *(with the classification* $TS : \{bio, chem\}$*), as Alice's security clearance is lower than the classification for* O_1*. However, Alice cannot write to the object* O_2 *(with the classification* $S : \{bio\}$*), as the* $sl(Alice)$ *is not less than or equal to* $sl(O_2)$*. Furthermore, Alice cannot write to* O_3 *(with the classification* $S : \{\}$*).*

Whenever a subject wants to access an object these two properties are verified prior to granting access. It is important to note that the equations are very similar, the only real difference is whether the subject dominates the object–simple security condition–or the object dominates the subject–star property.

Simple Public Key Infrastructure

"In access control of shared computing resources, the authorization problem addresses the following question: given a security policy, should a principal be allowed access to a specific resource" (Hermanns and Palsberg, 2006)? In the Simple Public Key Infrastructure (SPKI)[1], the security policy is given by a set of signed certificates, and proof of authorization consists of a set of certificate chains.

In SPKI, a *principal* can be any of the following: individual, system process, host, or other entities. All principals are identified with their public keys. The identity of a principal is established through the validation of the corresponding public key. $\mathscr{K}$ is used to denote the set of all principals and uses K, often with subscripts or superscripts (such as $K,\ KA,\ K_B,\ K'$, etc.), to denote a principal. An $identifier$ is a word (such as A or Bob) defined over some given standard alphabet.

The set of identifiers is denoted by $\mathscr{A}$. A *term* is a key that is followed by zero or more identifiers. A term can also be a local name or an extended name. A local name has the form KA, where $\mathscr{K} \in K$ and $\mathscr{A} \in A$, and an extended name is a principal followed by more than one identifier. A *local name* has the form KA where $K \in \mathscr{K}$ and $A \in \mathscr{A}$ (i.e. $KBob,\ KAlice$). An *extended name* is a key followed by more than one identifier, i.e. $K\ csusb\ ids\ faculty$.

SPKI has two types of certificates, or "certs":

- Name Certificates (name certs) define the names available in an issuer's local namespace. Specifically, a name cert provides a definition of a local name in the issuer's local namespace. Only key K may issue or sign a cert that defines a name in its local namespace. A name cert C is a signed 4-tuple $(K,\ A,\ S,\ V)$. The issuer

[1] an Internet Protocol Ellison et al., 1999. The current (2.0) version of SPKI is a merger of SPKI 1.0 and the Simple Distributed Security Infrastructure (SDSI) 1.0.

(principal) K is a public key and the certificate is signed by K. A is an identifier. The *subject* S is a term. S gives additional meaning for the local name KA. V is the validity specification of the certificate often taking on a time-span, i.e., the cert is valid from date/time x to y.

- Authorization Certificates (auth certs) grant authorizations (from an issuer to a subject), or delegate the ability to grant authorizations. An auth cert is a 5-tuple $(K,\ S,\ D,\ T,\ V)$ where K, S, and V are identical to the definition provided for name certs and play the same role. If the delegation flag D is set, then the subject receiving the authorization can delegate the authorization to any other key. The authorization specification T specifies the permission being granted; for example, it may specify the permission to read or write to a file or to grant login access to a particular host.

With SPKI, trust is decentralized; any principal may transfer a specific permission to another principal. A principal may transfer read and write or simply just write only privileges to a subject. Additionally, a principal may allow the subject to further transfer those permissions–referred to as delegation.

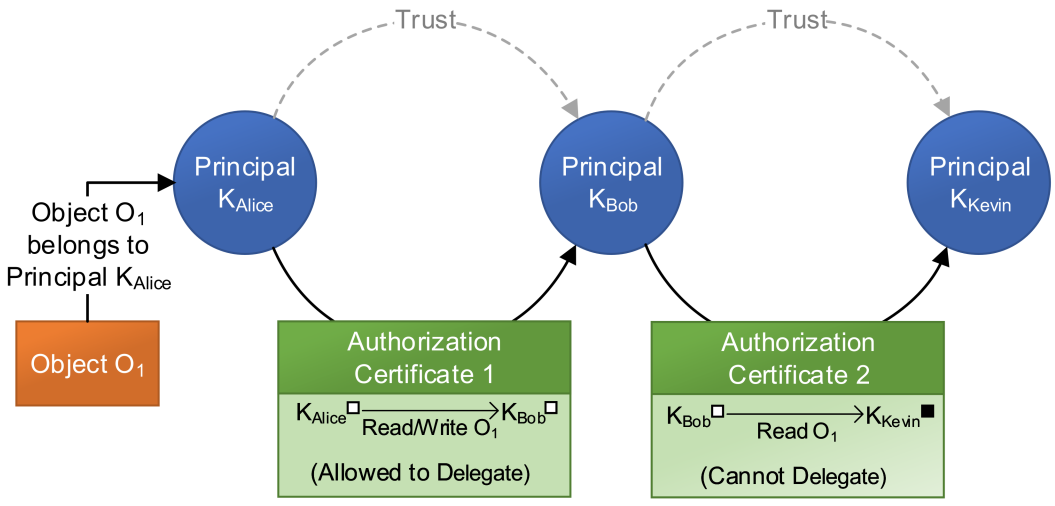


Figure 2.3: Example of SPKI authorization certificate and delegation.

In Figure 2.3, the first auth cert (AC1) has: the principal (K) as K_{Alice}, the subject (S) as K_{Bob}, the delegation flag (D) set to true and represented with an unfilled block, the authorization specification (T) as Read/Write O_1 and the validity (V) is omitted, meaning unlimited validity. The second auth cert (AC2) has: the principal (K) as K_{Bob}, the subject (S) as K_{Kevin}, the delegation flag (D) set to false and represented with a filled block, the authorization specification (T) as Read O_1, and the validity (V) is omitted.

Example 5. *K_{Alice} issues AC1 to K_{Bob} allowing K_{Bob} to read and write to O_1 along with the authority to delegate read and write access to O_1. K_{Bob} issues AC2 to K_{Kevin} allowing K_{Kevin} to only read O_1 and does not pass on the ability to delegate; meaning K_{Kevin} cannot transfer the permission to read O_1 any further. When K_{Kevin} attempts to access O_1, K_{Kevin} must present both certificates, AC1 and AC2, as proof.*

Active Bundle

Active Bundle (AB) is a mechanism used to protect the privacy of data and its user's identities (Othman and Lilien, 2009; Othman, 2010). An AB is a construct that bundles together three components sensitive data, metadata, and a virtual machine (VM). The three components are collectively called a *payload* of the bundle (Othman, 2010).

Sensitive data: is a digital content (*i.e.* bank wire transfer or patient personal data) that needs to be protected from privacy violations, data leaks, unauthorized accesses, etc.

Metadata: contains information about data carried by the AB. The metadata includes privacy and access control policies for sensitive data.

Virtual machine: controls and manages the program enclosed in an active bundle. The essential task of the virtual machine is the enforcement of the privacy and other policies specified by the metadata. It guarantees the appropriate access control for sensitive data of the bundle. For example, it discloses to the authorized entity (called

guardian) the proportion of sensitive data it is entitled to access. The VM also performs integrity checks for the bundle. The VM decides whether to enable a bundle or not; considering the trustworthiness of the host as the criterion.

The active bundle scheme protects sensitive data as long as they are accessed through an active bundle. Active bundle offers protection mechanisms (such as the implementation of disclosure rules, evaporation, and apoptosis).

The creator of an active bundle provides data to a bundle disseminator (middleware) that automatically generates metadata for the bundle. The creator uses a pseudo-random number generator *PRNG* to generate a random number N. Next, the creator calculates the hash $H(N)$ of N and sends $H(N)$ and *PRNG* to a secure server. The server generates the hash value $K = H(S,\ PRNG,\ H(N))$, and encryption $E_s(PRNG)$, where S is the server's private key. Then, the security server returns K and $E_s(PRNG)$ to the creator. The creator uses the key K as the encryption key, and an encryption algorithm EA (e.g., AES 128) to encrypt sensitive data and metadata. In addition, the creator loads the following information into the active bundle: $EA,\ E_S(PRNG)$, and VM. Only $E_S(P)$ is encrypted. EA is not encrypted since it is a public encryption algorithm. The VM is also not encrypted. Once the bundle is created and passed to an owner, the owner decides if the bundle should be publicly available or not. If so, the owner registers it in an active bundle directory. The bundle is now ready for dissemination to one or more hosts.

Bundle exchange is realized by using guardian buddies to transfer active bundles between guardians or to a destination. The source guardian (which could be the owner if it is the very first dissemination) forwards the active bundle to Buddy B_1, B_1 sends the bundle to Buddy B_2, and B_2 forwards the bundle to the destination host. Note that B_1 and B_2 are selected based on their trustworthiness.

Once the destination host receives an active bundle, it can enable or store the AB.

The host can then active bundle at any time. A prerequisite for enabling an active bundle, through its VM, is obtaining the necessary decryption key. The VM requests it from the security server. First, the VM sends $E_s(PRNG)$ (included in the bundle's metadata) to the security server. The server decrypts $E_s(PRNG)$ using its private key S. Using $PRNG$, the server generates a random number M, and then the hash value $K = H(S,\ PRNG,\ H(M))$. Finally, the server sends K to the VM of the active bundle.

Active bundles have been used to protect patients' electronic medical records (Salih, Lilien, and Othman, 2012). Active bundles are used to protect sensitive data outsourced to a cloud throughout their entire life-cycle both in the cloud as well as outside of the cloud (Sarhan and Carr, 2017). Implementation of an active bundle within a MLS environment would require a secured operating system such as Security-Enhanced Linux. Although physical implementation is outside the scope of this project, it is worth mentioning that Security-Enhanced Linux does support MLS.

Access Control Models in Multi-Domain Environments

Most of the research done on access and authorization control, for resource sharing within a multi-domain collaborative environments, have focused on frameworks based on Role-Based Access Control (RBAC) models (Crampton, 2002; Shehab, Bertino, and Ghafoor, 2005; Shafiq, 2006; Kim, Kim, Ryu, and Kim, 2009; Gouglidis, Mavridis, and Hu, 2013; Hilia, Chibani, Winter, and Djouani, 2017). One of the most referenced works on inter-operation and access control management, specifically for MLS multi-domain environments, was done by Dawson, Qian, and Samarati. They proposed three formal properties for correctly specifying the mapping between security levels of multiple domains with different security level lattices. The three properties proposed are consistency, non-ambiguity, and nonredundancy properties (Dawson, Qian, and Samarati, 2000). The consistency property states that the dominance relationship $l_i \geq l_j$ formed by a

security mapping ($\geq$) should not affect the original and local dominance relationship $l_i \geq_x l_j$ in the individual domain D_x:

Consistency Property *For all lattices* $\mathscr{L}_x = \langle L_x, \geq_x \rangle$, *and levels* $l_i, l_j \in L_x : l_i \geq l_j \Longrightarrow l_i \geq_x l_j$.

Note that the consistency property is the only constraint for ensuring that no unauthorized information flow will occur. Although ambiguity and redundancy in cross-lattice relationships do not cause any improper information flow, they would introduce unnecessary compilations and obscure translation. For this reason, the authors proposed two additional requirements, namely, nonambiguity property and nonredundancy property.

For the correctness of the security-level mapping, it is required that cross-lattice relationships must not be ambiguous. The nonambiguity property states that a security level l_j dominated by a security level l_i in the trusting domain cannot map to a security level that dominates a level to which l_i maps. Formally, it can be expressed as:

Nonambiguity Property *For any trusting domain* D_A *and trusted domain* D_S, *and levels* $l_i, l_j \in L_A, l_u, l_v \in L_S : (l_i \geq_A l_j) \wedge (l_u \geq_S l_v) \wedge (l_i \geq l_v) \in Map_A \Longrightarrow (l_j \geq l_u) \notin Map_A$.

The last property proposed is called the nonredundancy property. The nonredundancy property has two conditions that must be satisfied:

Nonredundancy Property

(i) *For all* $l_i, l_j \in L_A, l_u \in L_S : (l_i \geq_A l_j) \wedge (l_i \geq l_u) \in Map_A \Longrightarrow (l_j \geq l_u) \notin Map_A$.

(ii) *For all* $l_i \in L_A, l_u, l_v \in L_S : (l_u \geq_S l_v) \wedge (l_i \geq l_u) \in Map_A \Longrightarrow (l_i \geq l_v) \notin Map_A$.

Although the authors formally define the important properties, which can be used

as requirements for specifying the correct cross-lattice relationships between different domains, they did not discuss the effect on the domains. Specifically, the BLP violations related to the incorrect mapping. There appears to be no literature that addresses the BLP violations related to incorrect mapping. Therefore, Chapter 3 includes an investigation into information leakage within cross-lattice relationships along with the role trust plays when security levels are incorrectly mapped.

CHAPTER 3

MLS VIOLATIONS IN MULTI-DOMAIN ENVIRONMENTS

This chapter identifies several BLP security violations that arise when two domains map their MLS policies incorrectly between one another. Since the issues related to incorrect mapping have not been explored in regards to BLP violations, six test scenarios (Figure 3.1) are developed using a combination of the direction of trust (one-way, two-way) and their type of trust (partial read, partial write, and full read and write).

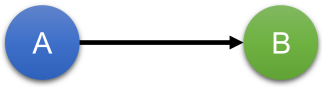
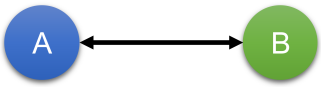
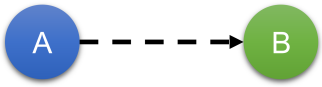
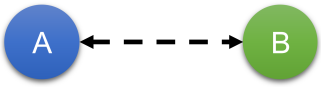
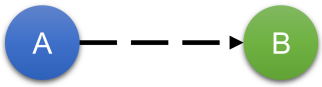
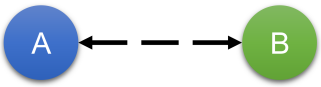
	One-way	**Two-way**
Full (Read/Write)	One-way full trust A trusts B ∴ B can read and write to A 	Two-way full trust A trusts B & B trusts A∴ B can read and write to A A can read and write to B 
Partial (Read)	One-way Partial (Read) A trust B ∴ B can read from A 	Two-way (Read) A trust B & B trusts A∴ B can read from A A can read from B 
Partial (Write)	One-way Partial (Write) A trust B ∴ B can write to A 	Two-way (Write) A trust B & B trusts A∴ B can write to A A can write to B 

Figure 3.1: Test Scenario Matrix.

Test scenarios

The following six test scenarios are built around two domains D_1 and D_2 (Figure 3.2), each with its own MLS structure. For simplicity, a lattice structure that does not include compartments is adopted. The correct security mapping would map D_1 *High* to D_2 *Top Secret* and D_1 *Low* to D_2 $Unclassified$.

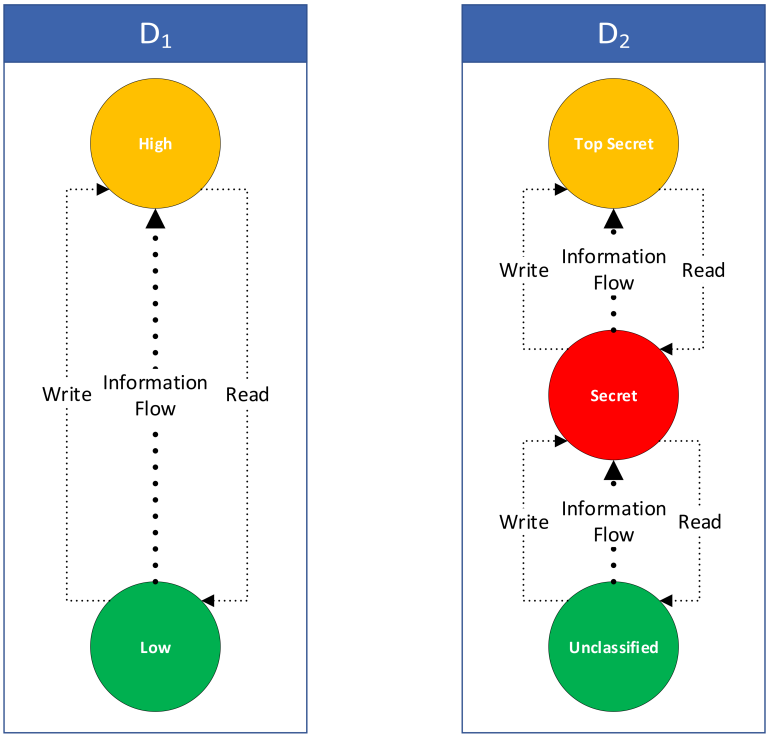


Figure 3.2: The two domains used for the six test scenarios.

Test Scenario 1 (one-way full trust)

The following test scenario depicts an incorrect mapping where D_1 *High* maps to D_2 *Unclassified* and D_1 *Low* maps to D_2 *Top Secret* (Figure 3.3). Because of this one-way full trust mapping:

- D_2 *Unclassified* can read from D_1 *High*
- D_2 *Unclassified* can write to D_1 *High*
- D_2 *Top Secret* can read from D_1 *Low*
- D_2 *Top Secret* can write to D_1 *Low*

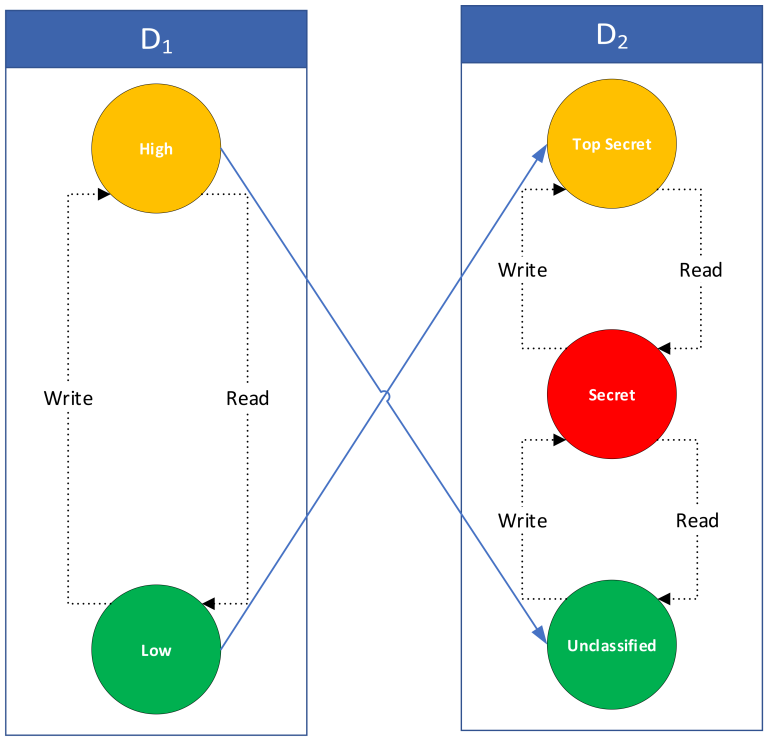


Figure 3.3: One-way full trust with incorrect mapping.

Star property violation The Star Property is violated (write down) in D_2 because the information is able to flow from D_2 *Top Secret* to D_2 $Unclassified$ (Figure 3.4).

$$D_2 \textit{ Top Secret writes to } D_1 \textit{ Low} \tag{3.1}$$

$$D_1 \textit{ Low writes to } D_1 \textit{ High} \tag{3.2}$$

$$D_2 \textit{ Unclassified reads from } D_1 \textit{ High} \tag{3.3}$$

Simple security condition violation The simple security condition is violated (read up) in D_1 because the information is able to flow from D_1 *High* to D_1 *Low* (Figure 3.5).

$$D_2 \textit{ Unclassified reads from } D_1 \textit{ High} \tag{3.1}$$

$$D_2 \textit{ Unclassified writes to } D_2 \textit{ Secret} \tag{3.2}$$

$$D_2 \textit{ Secret writes to } D_2 \textit{ Top Secret} \tag{3.3}$$

$$D_2 \textit{ Top Secret writes to } D_1 \textit{ Low} \tag{3.4}$$

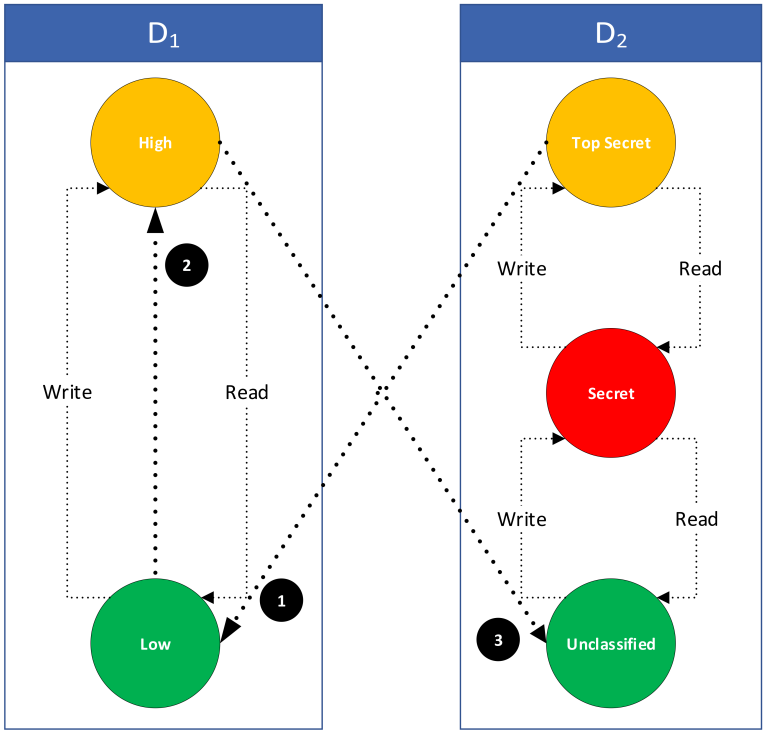


Figure 3.4: Start property violation.

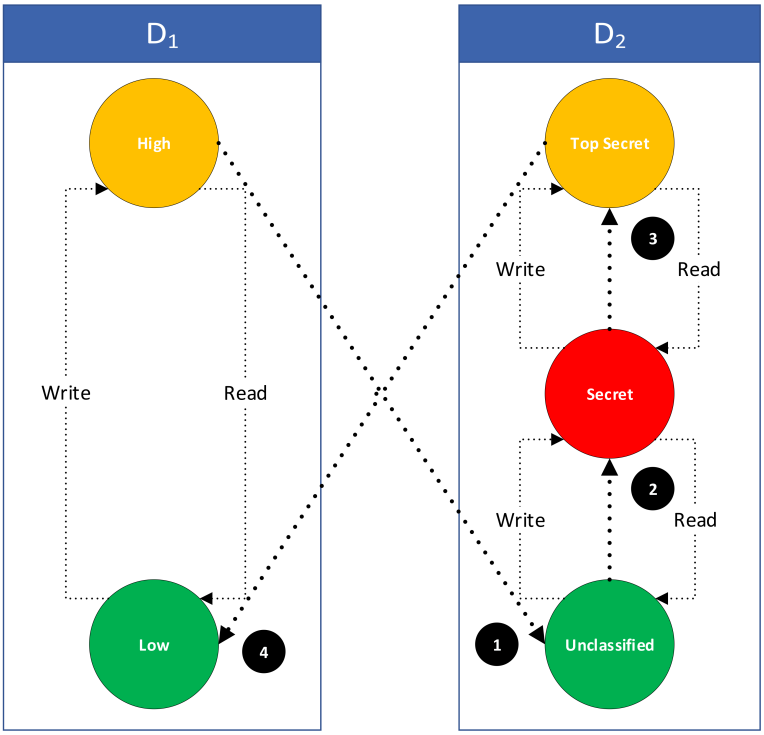


Figure 3.5: Simple security condition violation.

Test scenario 2 (two-way full trust)

The following test scenario depicts an incorrect mapping where D_1 *High* maps to D_2 *Unclassified* and D_1 *Low* maps to D_2 *Top Secret* (Figure 3.6). Because of this two-way full trust mapping:

- D_2 *Unclassified* can read from D_1 *High*
- D_2 *Unclassified* can write to D_1 *High*
- D_1 *High* can read from D_2 *Unclassified*
- D_1 *High* can write to D_2 *Unclassified*
- D_2 *Top Secret* can read from D_1 *Low*
- D_2 *Top Secret* can write to D_1 *Low*
- D_1 *Low* can read from D_2 *Top Secret*
- D_1 *Low* can write to D_2 *Top Secret*

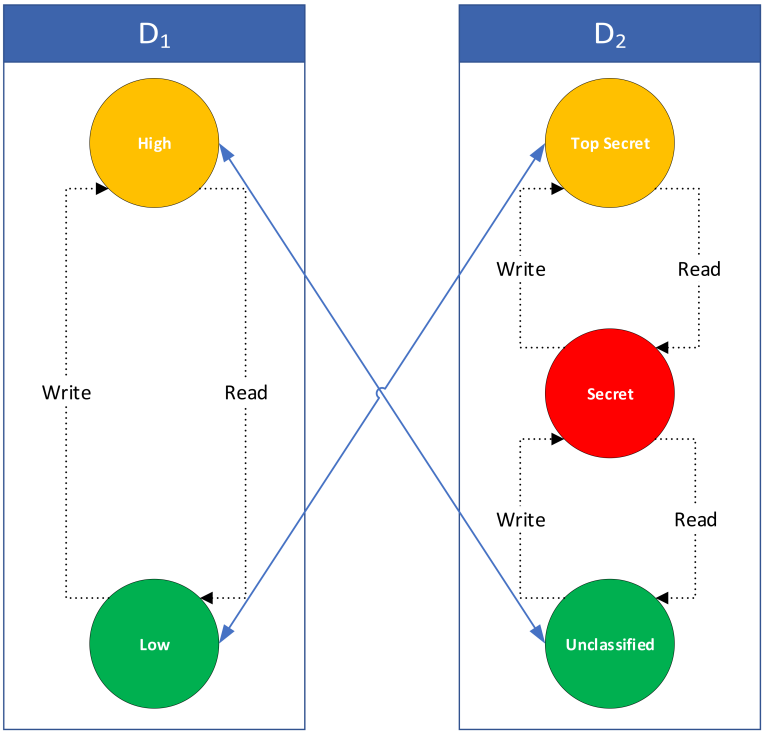


Figure 3.6: Two-way full trust with incorrect mapping.

Violations *Test scenario* 1 $\subseteq$ *Test scenario* 2, therefore, the same two violations that exist in Test scenario 1 exist in Test scenario 2. Moreover, Test scenario 2 will also contain the inverse violations, resulting in both simple security and star property violations in both D_1 and D_2.

Test scenario 3 (one-way partial trust to read)

The following test scenario depicts an incorrect mapping where D_1 *High* maps to D_2 *Unclassified* (Figure 3.7). The mapping of D_1 *Low* to D_2 *Top Secret* is of no concern as D_2 *Top Secret* reading D_1 *Low* is valid. Because of this one-way partial trust to read mapping:

- D_2 *Unclassified* can read from D_1 *High*
- D_2 *Top Secret* can read from D_1 *Low*

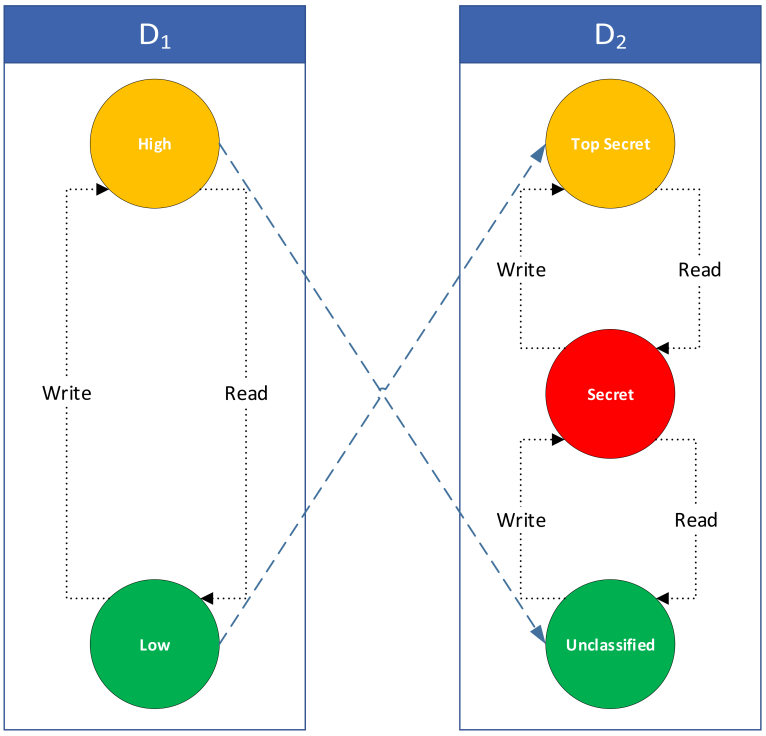


Figure 3.7: One-way full partial trust to read with incorrect mapping.

Simple security condition violation The MLS policies in D_1 are not violated in D_1, nor are the MLS policies of D_2 violated within D_2. However, the MLS policies of D_1 are not upheld by D_2. The simple security condition is violated (read up) as both D_2 *Unclassified* and D_2 *Secret* are able to read D_1 *High* (Figure 3.8).

$$D_2 \; Unclassified \; reads \; from \; D_1 \; High \tag{3.1}$$

$$D_2 \; Unclassified \; writes \; to \; D_1 \; Secret \tag{3.2}$$

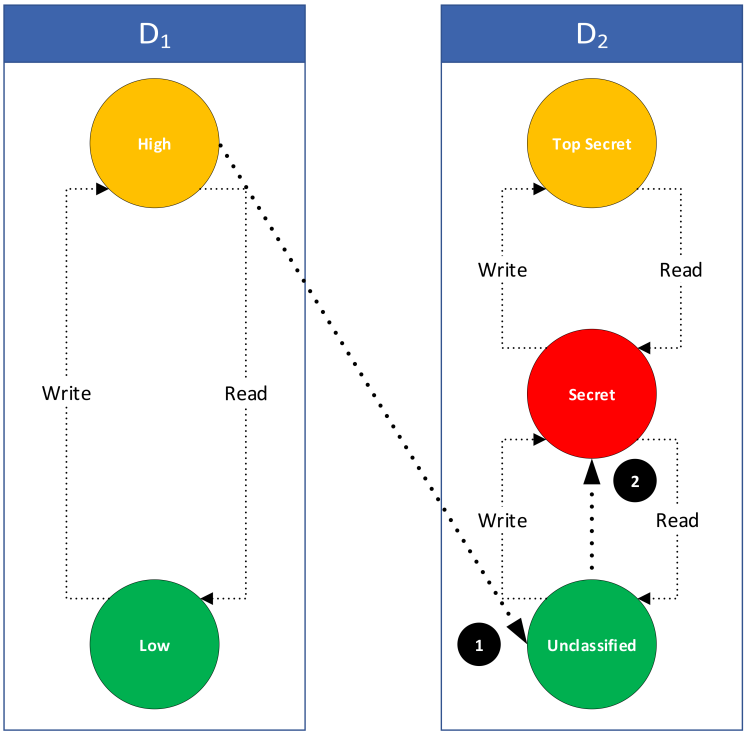


Figure 3.8: Simple security condition violation.

Test scenario 4 (one-way partial trust to write)

The following test scenario depicts an incorrect mapping where D_1 *Low* maps to D_2 *Top Secret* (Figure 3.7). The mapping of D_1 *High* to D_2 $Unclassified$ is of no concern, as D_2 $Unclassified$ writing to D_1 *High* is valid. Because of this one-way partial trust to read mapping:

- D_2 $Unclassified$ can write to D_1 *High*
- D_2 *Top Secret* can write to D_1 *Low*

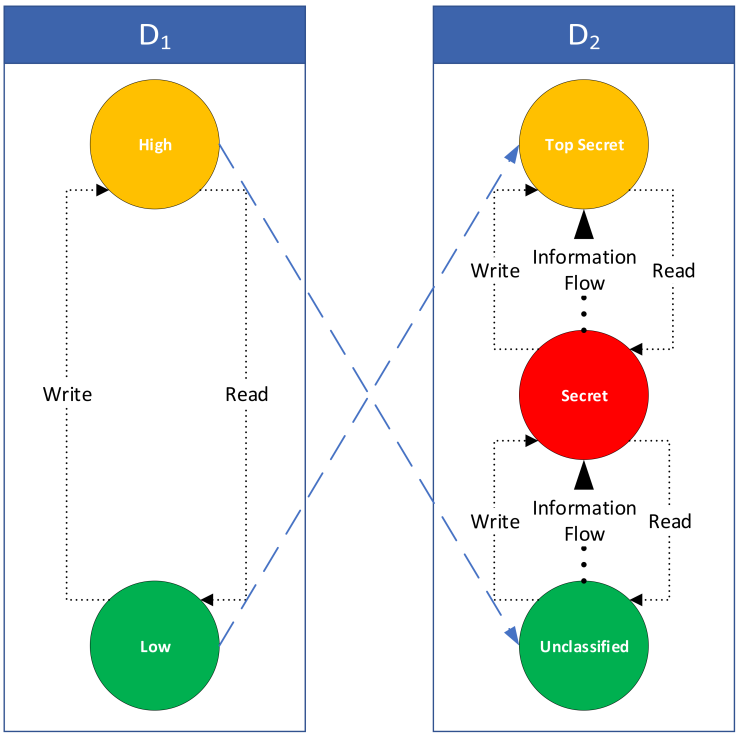


Figure 3.9: One-way full partial trust to write with incorrect mapping.

Star property violation The MLS policies in D_1 are not violated in D_1, nor are the MLS policies of D_2 violated within D_2. However, the MLS policies of D_2 are not upheld by D_1. The star property is violated (write down) as D_2 *Top Secret* is able to write to D_1 *Low* (Figure 3.10).

$$D_2 \textit{ Top Secret writes to } D_1 \textit{ Low} \tag{3.1}$$

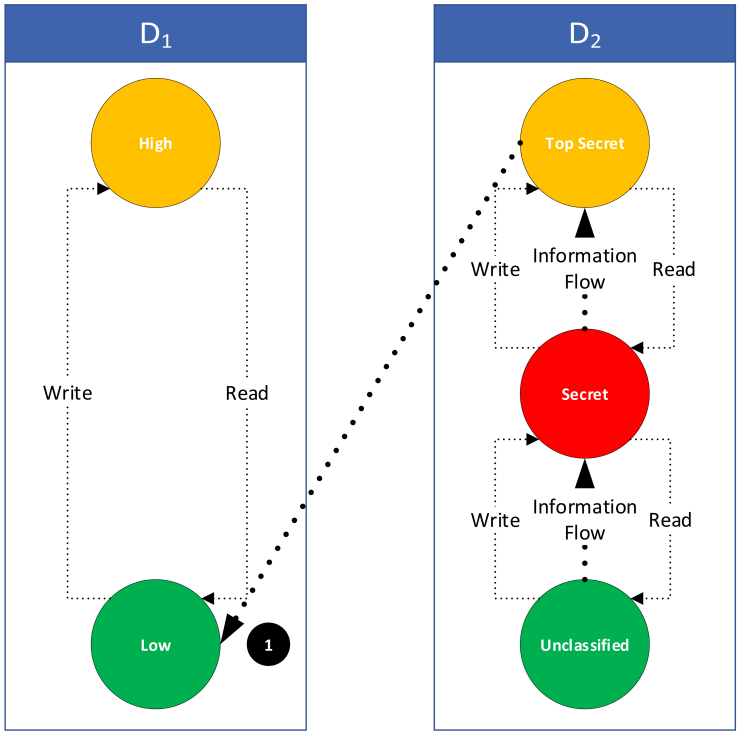


Figure 3.10: Star property violation.

Test scenario 5 (two-way partial trust to read)

The following test scenario depicts an incorrect mapping where D_1 *High* maps to D_2 *Unclassified*, and D_1 *Low* maps to D_2 *Top Secret* (Figure 3.11). Because of this two-way partial trust to read mapping:

- D_2 *Unclassified* can read from D_1 *High*
- D_1 *High* can read from D_2 *Unclassified*
- D_2 *Top Secret* can read from D_1 *Low*
- D_1 *Low* can read from D_2 *Top Secret*

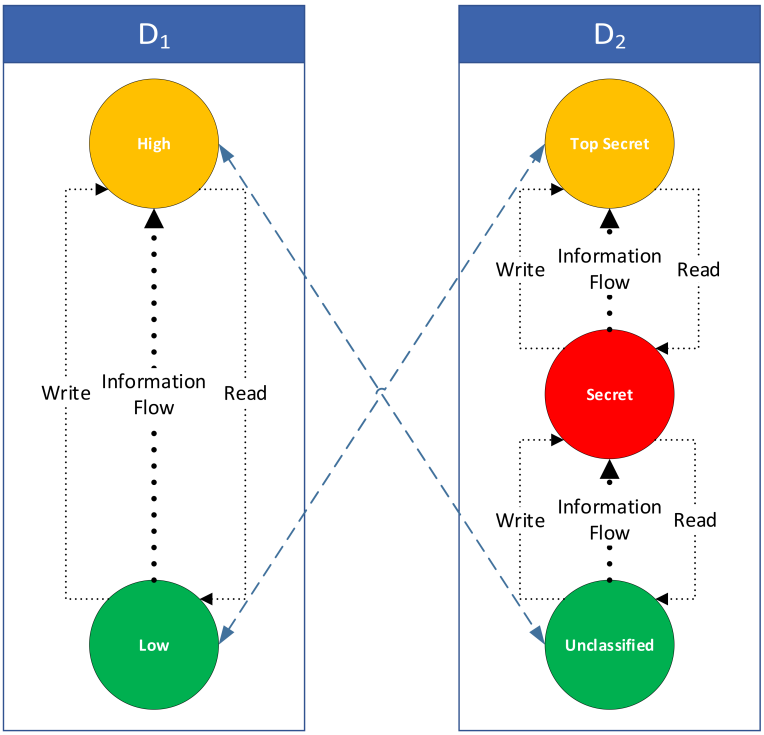


Figure 3.11: Two-way partial trust to read with incorrect mapping.

Violations *Test scenario* 3 $\subseteq$ *Test scenario* 5, therefore, the same violations that exist in Test scenario 3 exist in Test scenario 5.

Simple security condition violation The simple security condition is violated (read up) in D_1 because the information is able to flow from D_1 *High* to D_1 *Low* (Figure 3.12).

$$D_2\ Unclassified\ reads\ from\ D_1\ High \tag{3.1}$$

$$D_2\ Secret\ reads\ from\ D_2\ Unclassified \tag{3.2}$$

$$D_2\ Top\ Secret\ reads\ from\ D_2\ Secret \tag{3.3}$$

$$D_1\ Low\ reads\ from\ D_2\ Top\ Secret \tag{3.4}$$

Simple security condition violation The simple security condition is violated (read up) in D_2 because the information is able to flow from D_2 *Top Secret* to D_2 *Secret* and D_2 *Unclassified* (Figure 3.13).

$$D_1\ Low\ reads\ from\ D_2\ Top\ Secret \tag{3.1}$$

$$D_1\ High\ reads\ from\ D_1\ Low \tag{3.2}$$

$$D_2\ Unclassified\ reads\ from\ D_1\ High \tag{3.3}$$

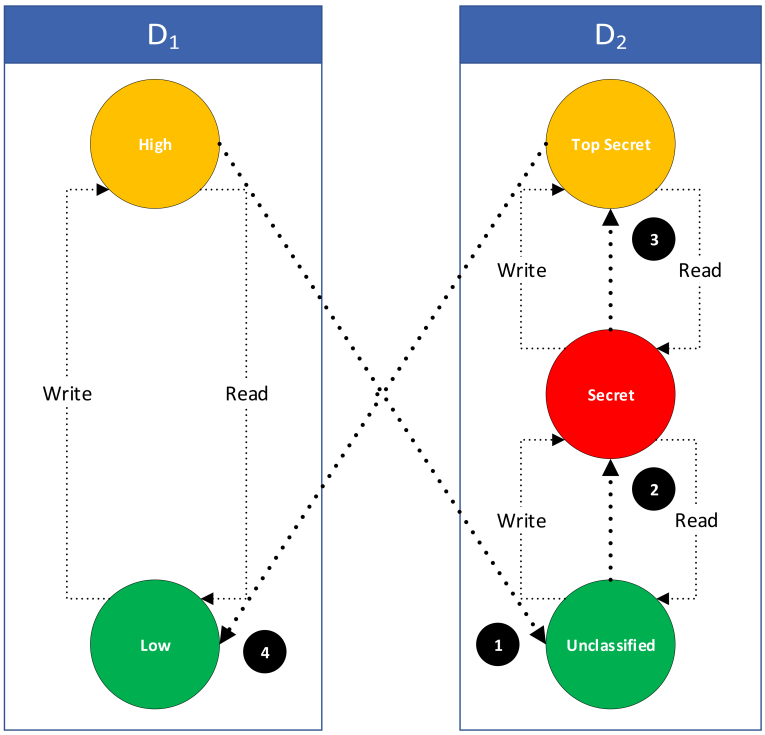


Figure 3.12: Simple security condition violation.

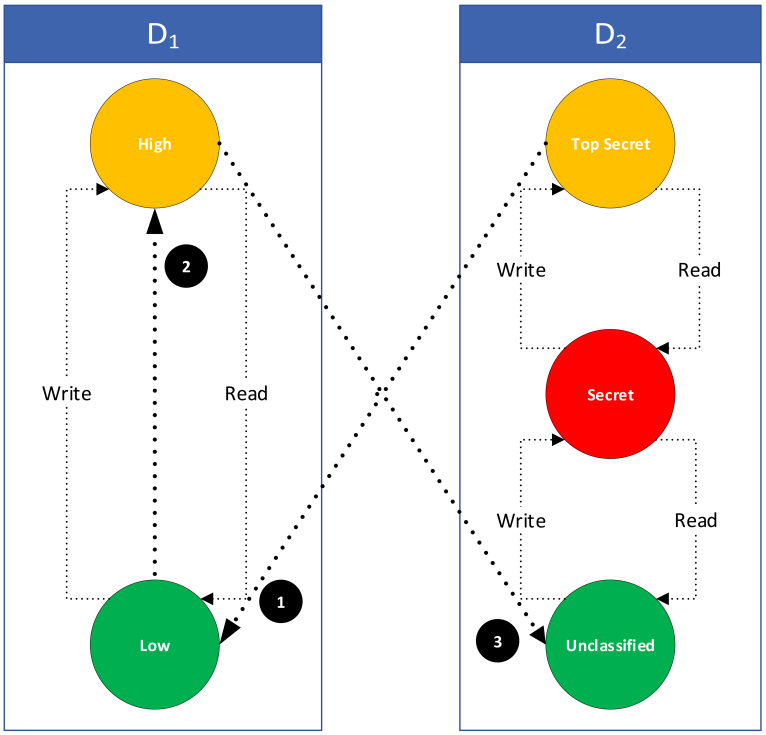


Figure 3.13: Simple security condition violation.

Test scenario 6 (two-way partial trust to write)

The following test scenario depicts an incorrect mapping where D_1 *High* maps to D_2 *Unclassified*, and D_1 *Low* maps to D_2 *Top Secret* (Figure 3.14). Because of this two-way partial trust to write mapping:

- D_2 *Unclassified* can write to D_1 *High*
- D_1 *High* can write to D_2 *Unclassified*
- D_2 *Top Secret* can write to D_1 *Low*
- D_1 *Low* can write to D_2 *Top Secret*

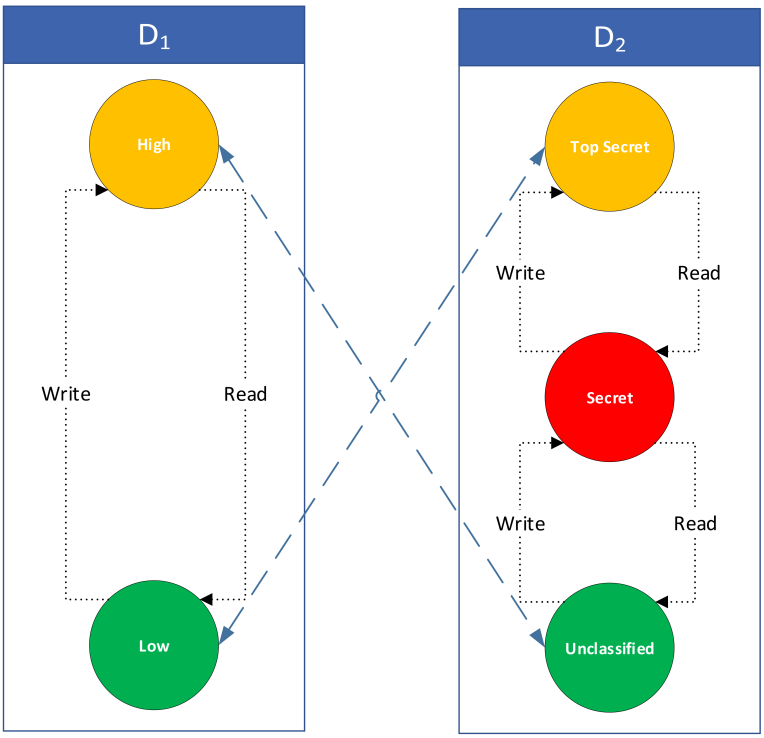


Figure 3.14: Two-way partial trust to write with incorrect mapping.

Violations *Test scenario* 4 $\subseteq$ *Test scenario* 6, therefore, the same violations that exist in Test scenario 4 exist in Test scenario 6.

Star property violation The star property is violated (write down) in D_1 because the information is able to flow from D_1 *High* to D_1 *Low* (Figure 3.15).

$$D_1 \; High \; writes \; to \; D_2 \; Unclassified \tag{3.1}$$

$$D_2 \; Unclassified \; writes \; to \; D_2 \; Secret \tag{3.2}$$

$$D_2 \; Secret \; writes \; to \; D_1 \; Top \; Secret \tag{3.3}$$

$$D_2 \; Top \; Secret \; writes \; to \; D_1 \; Low \tag{3.4}$$

Star property violation The star property is violated (write down) in D_2 because the information is able to flow from D_2 *Top Secret* to D_2 *Secret* and D_2 *Unclassified* (Figure 3.16).

$$D_2 \; Top \; Secret \; writes \; to \; D_1 \; Low \tag{3.1}$$

$$D_1 \; Low \; writes \; to \; D_1 \; High \tag{3.2}$$

$$D_1 \; High \; writes \; to \; D_2 \; Unclassified \tag{3.3}$$

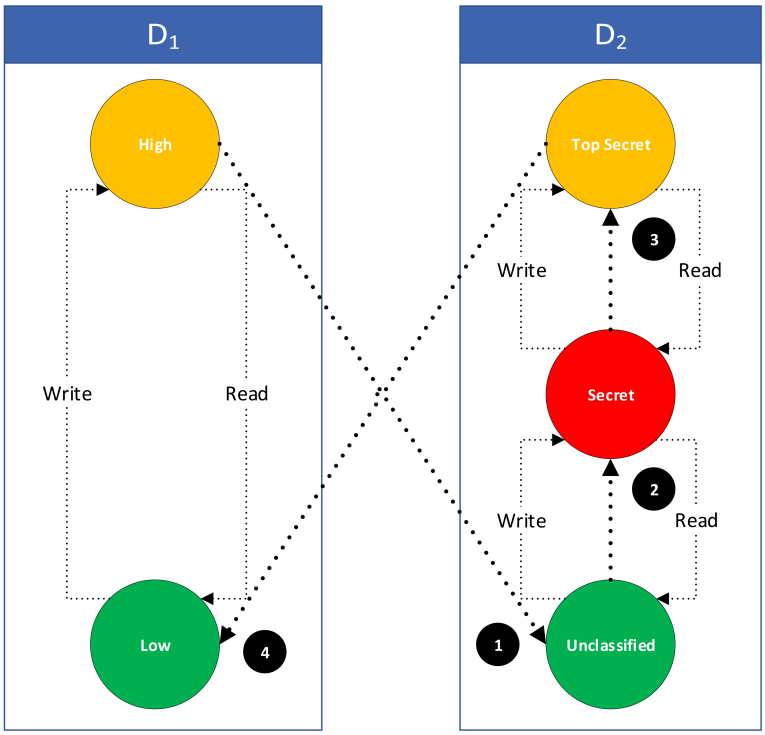


Figure 3.15: Simple security condition violation.

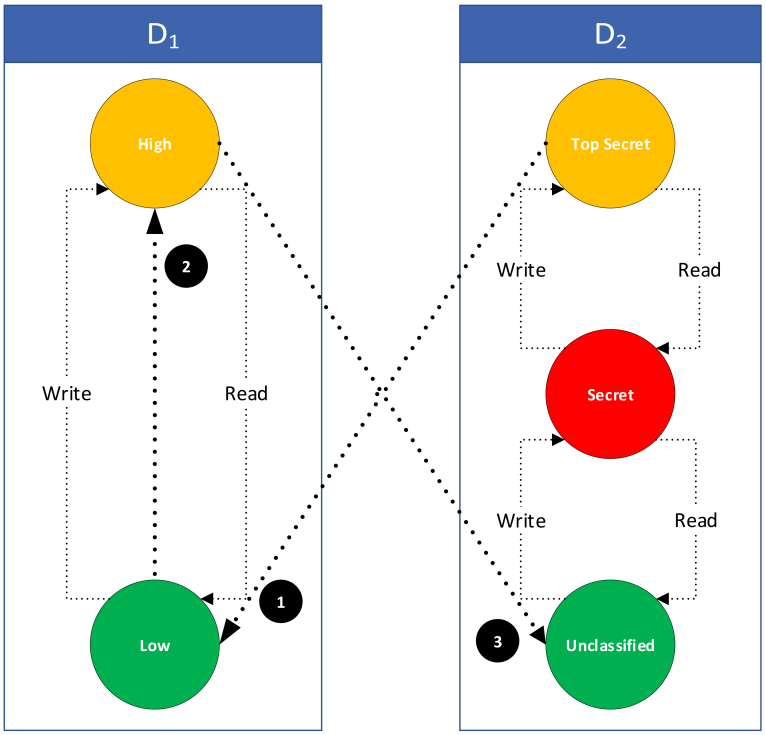


Figure 3.16: Simple security condition violation.

Violations Matrix

Test Scenario	**D1 Violation**	**D2 Violation**	**MLS Policy not Upheld**
(1) One-way Full Trust	Simple Security	Start Property	Trustor (D1) Trustee (D2)
(2) Two-way Full Trust	Simple Security Start Property	Simple Security Start Property	Trustor (D1) Trustee (D2)
(3) One-way Partial Trust to Read			Trustor (D1)
(4) One-way Partial Trust to Write			Trustee (D2)
(5) Two-way Partial Trust to Read	Simple Security	Simple Security	Trustor (D1) Trustee (D2)
(6) Two-way Partial Trust to Write	Start Property	Start Property	Trustor (D1) Trustee (D2)

Table 3.1: Violations matrix

The direction of the trust provides minimal security benefit to either domain. The only test scenario that provides any protection to the trustor is test scenario 4, where the trustor gives the trustee write-only privileges. Furthermore, the only protection to the trustee is in test Scenario 3, where the trustor gives the trustee read-only privileges. When two domains create a trust relationship, the direction of trust and the type of trust does not provide protection for both parties. The incorrect mapping between the domains can cause issues for both domains. At best, one of the domains will have their MLS policy violated.

The six test scenarios are not exhaustive. However, the scenarios highlight the need for domains to understand the MLS policies of any domain they interact with and trust. Lacking understanding of the MLS policies of trusted domains can lead to various violations resulting in information leakage.

CHAPTER 4

BLP MODEL IN SPKI

Logical Representation of BLP in SPKI

There are two ways to implement the lattice in Figure 2.1 (reproduced here as Figure 4.1 for reader clarity). The first approach consists of a central MLS authority in a domain responsible for labeling all the objects and subjects. The second approach consists of a hierarchy of MLS authorities at each security level. The first approach is used in this project.

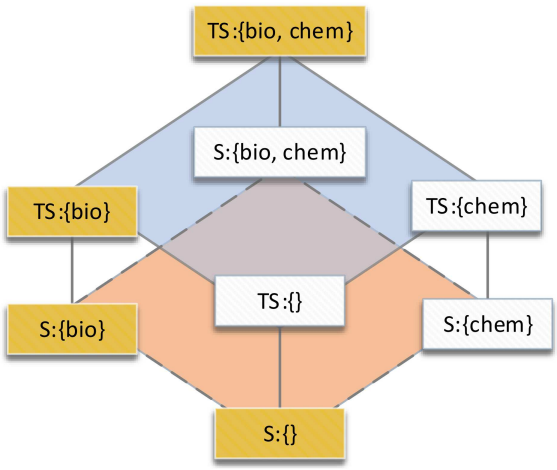


Figure 4.1: Lattice structure

In a local domain d, K_d is a domain controller and K_{MLS_d} is a sub-controller or security entity responsible for assigning the security levels or labels to the subjects and objects in domain d. For example, Alice and the object O_1 are labeled as having $TS : \{bio, chem\}$ and $S : \{\}$, respectively. This can be expressed by *SPKI* name cert as:

$$K_d \quad MLS\_Policy \longrightarrow K_{MLS\_d} \tag{4.1}$$

$$K_{MLS\_d} \quad TS:\{bio, chem\} \longrightarrow K_{Alice} \tag{4.2}$$

$$K_{MLS\_d} \quad S:\{\} \longrightarrow K_{O_1} \tag{4.3}$$

Both the simple and star property can be modeled in *SPKI* authorization certificates. For example, the simple property of the highlighted area of the MLS lattice shown in Figure 4.1 can be expressed as:

$$K_{MLS\_d}\ \square \xrightarrow[\text{read } O_{S:\{\}}]{} K_{MLS\_d} \quad S:\{\}\ \blacksquare \tag{4.1}$$

$$K_{MLS\_d}\ \square \xrightarrow[\text{read } O_{S:\{\}}]{} K_{MLS\_d} \quad S:\{bio\}\ \blacksquare \tag{4.2}$$

$$K_{MLS\_d}\ \square \xrightarrow[\text{read } O_{S:\{\}}]{} K_{MLS\_d} \quad TS:\{bio\}\ \blacksquare \tag{4.3}$$

$$K_{MLS\_d}\ \square \xrightarrow[\text{read } O_{S:\{\}}]{} K_{MLS\_d} \quad TS:\{bio\}\ \blacksquare \tag{4.4}$$

$$K_{MLS\_d}\ \square \xrightarrow[\text{read } O_{S:\{\}}]{} K_{MLS\_d} \quad TS:\{bio,\ chem\}\ \blacksquare \tag{4.5}$$

$$K_{MLS\_d}\ \square \xrightarrow[\text{read } O_{S:\{bio\}}]{} K_{MLS\_d} \quad S:\{bio\}\ \blacksquare \tag{4.6}$$

$$K_{MLS\_d}\ \square \xrightarrow[\text{read } O_{S:\{bio\}}]{} K_{MLS\_d} \quad TS:\{bio\}\ \blacksquare \tag{4.7}$$

$$K_{MLS\_d}\ \square \xrightarrow[\text{read } O_{S:\{bio\}}]{} K_{MLS\_d} \quad TS:\{bio,\ chem\}\ \blacksquare \tag{4.8}$$

$$K_{MLS\_d}\ \square \xrightarrow[\text{read } O_{TS:\{bio\}}]{} K_{MLS\_d} \quad TS:\{bio\}\ \blacksquare \tag{4.9}$$

$$K_{MLS\_d}\ \square \xrightarrow[\text{read } O_{TS:\{bio\}}]{} K_{MLS\_d} \quad TS:\{bio,\ chem\}\ \blacksquare \tag{4.10}$$

$$K_{MLS\_d}\ \square \xrightarrow[\text{read } O_{TS:\{bio,chem\}}]{} K_{MLS\_d} \quad TS:\{bio,\ chem\}\ \blacksquare \tag{4.11}$$

If Alice (with $TS : \{bio\}$ security clearance) sends a read request to an object $O_{S:\{\}}$ (whose security classification level is $S : \{\}$), she has to provide the following certifications to the MLS domain controller K_{MLS_d}:

$$K_{MLS\_d} \ \square \xrightarrow[\text{read } O_{S:\{\}}]{} K_{MLS\_d} \quad TS : \{bio\} \ \blacksquare \tag{4.1}$$

$$K_{MLS\_d} \quad TS : \{bio\} \ \longrightarrow \ K_{Alice} \tag{4.2}$$

The reference monitor or guard goes through the following verification process:

$$K_{MLS\_d} \ \square \xrightarrow[\text{read } O_{S:\{\}}]{} K_{MLS\_d} \quad TS : \{bio\} \ \longrightarrow \ K_{Alice}$$

Prolog Implementation

In order to test the mechanics of embedding MLS policies within SPKI, the Prolog program in Appendix C was developed to demonstrate both restricted and unrestricted dissemination. The program implements the relationships between domains found in Figure 4.2. In Figure 4.2 the solid black line represents unrestricted dissemination, the solid red line represents restricted dissemination, while the dashed black line represents unrestricted dissemination with an upstream restricted dissemination relationship.

Program Overview

The program first describes each object within the AB (O_1, O_2, and O_3) along with their respective security levels (*Low*, *Medium*, *High*) in lines 2 through 4. Following the declaration of the objects, lines 6 through 17 identify the various subjects, their security levels, and the domain that issued the certificate. Next, lines 20 through 33 define the authorization policy for the AB. Following the authorization policy is the trust policy in lines 35 through 48. Note that lines 35 and 36 represent a restrictive

dissemination certificate, where the certificate is only valid if the issuer is the Army. Lines 48 through 52 represent the recursive certificates, with line 48 handling the certificates with restrictive dissemination. Finally, lines 54 through 58 verify the certificate chain. The remaining lines are simply there to make the program more user-friendly by allowing the user to list the authorization policy, subjects, objects, and trust certificates.

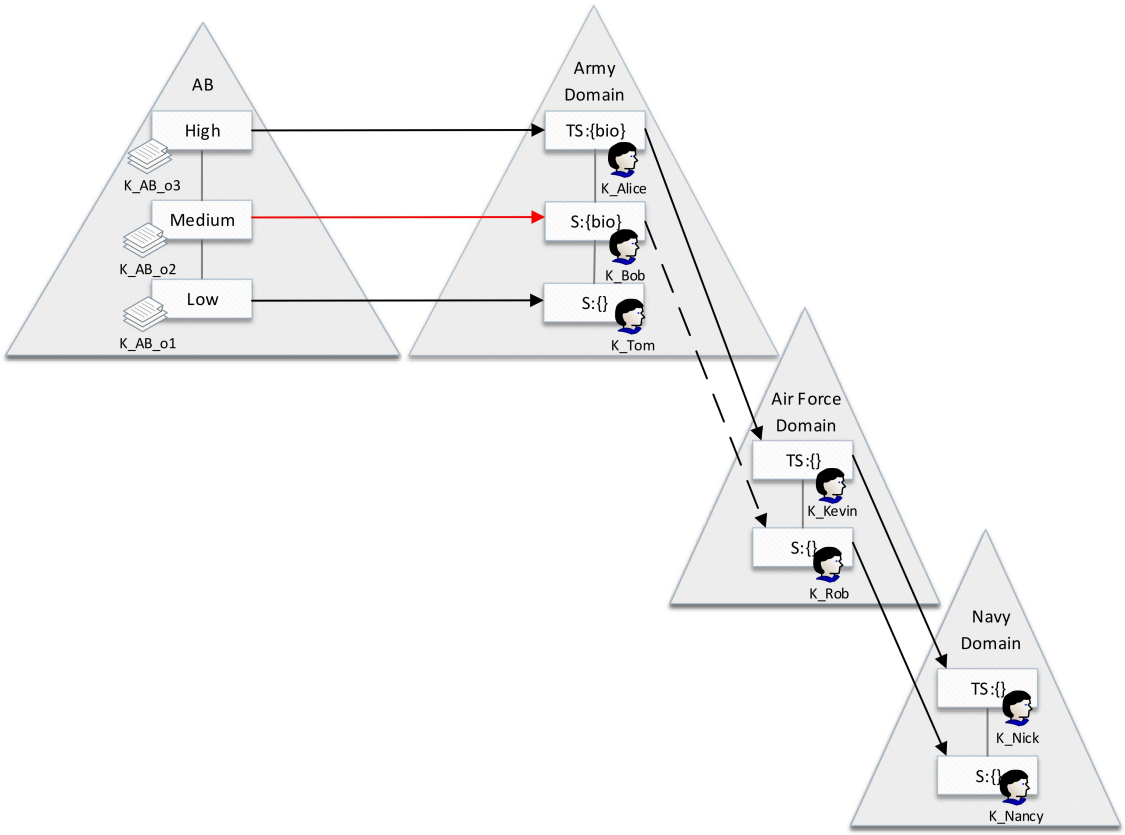


Figure 4.2: Prolog Example: Domain Trust

Program Examples

Example 6. *The following is an actual output of the Prolog terminal. The terminal shows the access K_{Alice} has, with respect to the objects (O_1, O_2, and O_3). Prolog was asked to display the access Alice has, and according to the program, Alice can read all three objects but only write to O_3.*

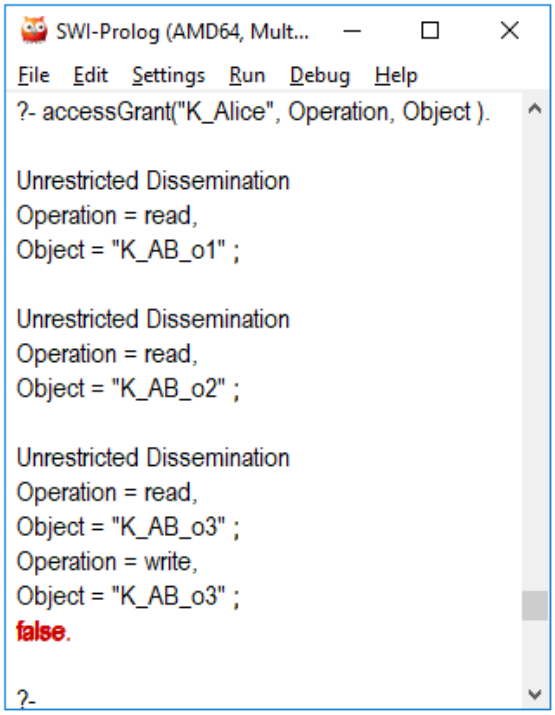


Figure 4.3: Prolog Example: Terminal output

Example 7. *The following is an example output and shows that K_{Kevin} from the Air Force Domain has the same privileges as K_{Alice} and the certificate chain is Unrestricted Dissemination. Note, that this is not an exact output, as the actual output would repeat the Unrestricted Dissemination line several times as it goes through the recursive process, it has been omitted and will be omitted in the following examples for simplicity.*

?- accessGrant("K_Kevin", read, "K_AB_o1").

Unrestricted Dissemination

true

?- accessGrant("K_Kevin", read, "K_AB_o2").

Unrestricted Dissemination

true

?- accessGrant("K_Kevin", read, "K_AB_o3").

Unrestricted Dissemination

true

?- accessGrant("K_Kevin", write, "K_AB_o3").

Unrestricted Dissemination

true

?- accessGrant("K_Kevin", write, "K_AB_o2").

Unrestricted Dissemination

false

?- accessGrant("K_Kevin", write, "K_AB_o1").

Unrestricted Dissemination

false

Example 8. *The following is an example output below and shows that K_{Bob} can read O_2 and O_1. In addition, K_{Bob} can write to O_2 and O_3. Also, note that the certificate chain is now Restricted Dissemination.*

?- accessGrant("K_Bob", read, "K_AB_o1").

Restricted Dissemination

true

?- accessGrant("K_Bob", read, "K_AB_o2").

Restricted Dissemination

true

?- accessGrant("K_Bob", read, "K_AB_o3").

Restricted Dissemination

false

?- accessGrant("K_Bob", write, "K_AB_o3").

Restricted Dissemination

true

?- accessGrant("K_Bob", write, "K_AB_o2").

Restricted Dissemination

true

?- accessGrant("K_Bob", write, "K_AB_o1").

Restricted Dissemination

false

Example 9. *The following is an example output below and shows that K_{Rob} cannot read or write to any object, despite the fact that K_{Rob} has a trust certificate from the Army, due to a restricted dissemination certificate in the chain.*

?- accessGrant(”K_Rob”, read, ”K_AB_o1”).

Restricted Dissemination

false

?- accessGrant(”K_Rob”, read, ”K_AB_o2”).

Restricted Dissemination

false

?- accessGrant(”K_Rob”, read, ”K_AB_o3”).

Restricted Dissemination

false

?- accessGrant(”K_Rob”, write, ”K_AB_o3”).

Restricted Dissemination

false

?- accessGrant(”K_Rob”, write, ”K_AB_o2”).

Restricted Dissemination

false

?- accessGrant(”K_Rob”, write, ”K_AB_o1”).

Restricted Dissemination

false

CHAPTER 5

MLS CROSS-DOMAIN SECURITY LEVEL MAPPING

In a MLS distributed environment, if one domain (trusting domain) D_i trusts another domain (trusted domain) D_j, both domains can build a cross-lattice relationship. The cross-relationship specifies which security levels of the trusting domain dominates (or is mapped to) which security levels of the trusted domain. This security level mapping between two domains is denoted as $l_i \geq l_j$, where $l_i \in L_i$, $l_j \in L_j$. Specifically, $\geq$ is used to represent the following:

- A dominance relationship holding in an individual lattice $\geq_x$ can be replaced.
- A cross-dominance or cross-lattice relationship from the trusting domain D_i to the trusted domain D_j, namely, $l_i \geq l_j$, where $l_i \in L_i$ and $l_j \in L_j$.
- A combination of the cross-lattice and (individual) lattice relationship.

A security level mapping $l_i \geq l_j$ can be implemented by the following SPKI name certificate: $l_i \longrightarrow l_j$. This certificate issued by D_i states that security level l_i of domain D_i is mapped to the security level l_j of D_j (in SPKI terms, the security level l_i is redefined as another security level l_j). In this project, both notations are used interchangeably. Note that security level mapping in a cross-lattice relationship are unidirectional from the trusting domain to the trusted domain.

Example 10. *Assume that there are two domains, the AB (5.1) and the Army domain (5.2) and the AB trusts the Army. The AB domain has three security levels:* $TS:\{bio\}$, $TS:\{\}$, *and* $Secret:\{\}$ *(5.3). The Army has the following security labels:* $TS:\{\}$, $Secret:\{\}$, *and* $Classified:\{\}$ *(5.4). If the security level* $K_{MLS\_AB}\ TS:\{bio\}$ *of the AB dominates the security level* $K_{MLS\_Army}\ TS\{\}$ *of the Army domain, we can express the cross-domain dominance relationship ((5.5)).*

$$D_{AB} \tag{5.1}$$

$$D_{Army} \tag{5.2}$$

$$L_{AB} = \{K_{MLS\_AB}\ TS:\{bio\},\ K_{MLS\_AB}\ TS:\{\},\ K_{MLS\_AB}\ Secret:\{\}\} \tag{5.3}$$

$$L_{Army} = \{K_{MLS\_Army}\ TS,\ K_{MLS\_Army}\ Secret:\{\},\ K_{MLS\_Army}\ Classified:\{\}\} \tag{5.4}$$

$$K_{MLS\_AB}\ TS:\{bio\} \geq K_{MLS\_Army}\ TS\{\} \tag{5.5}$$

This cross lattice relationship can be realized by the following name certificate:

$$K_{MLS\_AB}\ TS:\{bio\} \longrightarrow K_{MLS\_Army}\ TS$$

The above name certificate clearly shows that there is a cross security level mapping from security level $TS:bio$ *of the active bundle (trusting domain) to the security level* TS *of the Army domain (trusted domain).*

CHAPTER 6

MLS DATA DISSEMINATION

MLS Data Dissemination Strategy

Normally, the data disseminator (sender) must know the recipients prior to sending any data; furthermore, the sender must be aware of the recipient's security clearance prior to distribution. Knowledge of all recipients and their respective clearances is not always practical or even possible. By leveraging AB, SPKI, and a directory server, data can be disseminated prior to the identification of all recipients and their respective security clearances.

The sensitive data within the AB is secured through encryption. Once the AB is enabled, it's protected and controlled by the VM. The authorization policy and trust policy are defined within the metadata of the AB. The policies limit access to trusted entities. The AB signs both policies via SPKI, thus producing the authorization certificates and name certificates that are stored in the metadata unencrypted. Through Evaporation (partial self-destruction) and Apoptosis (full self-destruction), the AB can protect the sensitive data, allowing for the distribution of data without the fear of information leaking.

There are two main ways a domain can learn about the authorization and trust policies of an AB. The first way is to directly read both SPKI authorization certificates and name certificates from the metadata of an AB when it arrives at the domain for execution. Another way is through a reliable directory service. An AB publishes its SPKI authorization and name certificates to a directory server, thus allowing a third party to query the directory server in order to understand the authorization and trust

policies necessary to access a particular AB. The directory server acts as a global catalog allowing AB creators and consumers a secure pathway to distribute MLS data. Although the information within the directory server is public, restricting access and monitoring requests to and from the directory server would provide additional security. However, the mechanism of providing a reliable and secure directory service is beyond the scope of this project.

SPKI provides the mechanism by which the authorization and trust policies are defined and trusted, as each request made to the AB must be proven through a certificate chain. Additionally, SPKI allows for a distributed trust network, accomplished by issuing name certificates that represent trust. Therefore, the requester is not required to have a direct trust relationship with the AB; instead, trust can be transitive, if allowed by the AB.

MLS Data Dissemination Methods

There are two primary dissemination scenarios that must be supported, unrestricted dissemination and restricted dissemination. When a domain (*A*) creates a trust relationship with another domain (*B*), domain *A* must decide if domain *B* will be permitted delegation rights to extend the trust in a transitive manner. If domain *B* is permitted to delegate the trust to another domain (*C*), it is referred to as unrestricted dissemination. However, if domain *B* was disallowed from extending the trust to domain *C* it would be referred to as restricted dissemination.

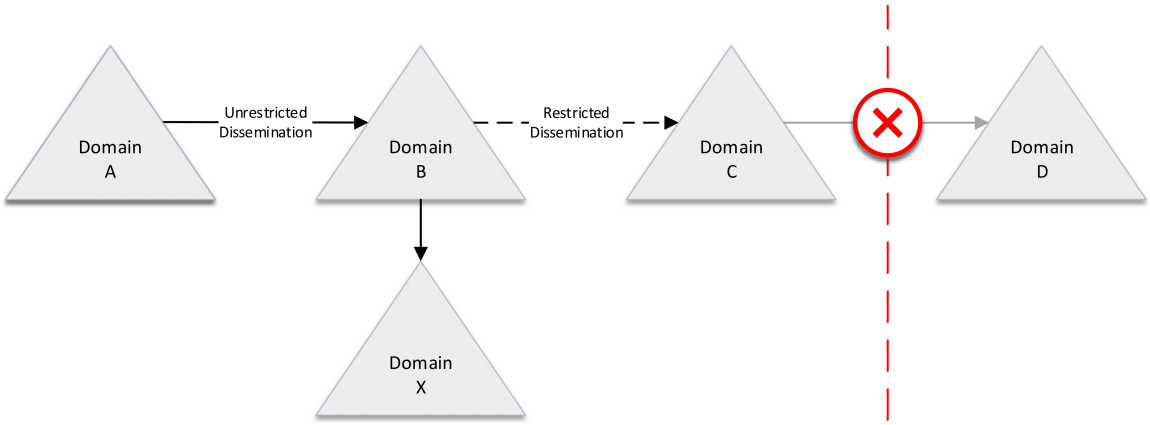


Figure 6.1: Dissemination scenario (Domain D is not trusted by Domain A).

Example 11. *In Figure 6.1, domain A provides domain B unrestricted dissemination rights, thereby allowing domain B the ability to further extend the trust relationship. As such, domain B issues unrestricted dissemination rights to domain X and restricted dissemination rights to domain C. The right to extend the trust can be limited anywhere down the line, in this example domain C cannot extend the trust any further. Although domain C has a trust relationship with domain D, domain D will not be trusted by domain A, whereas domain X would be allowed to extend.*

A more specific example can be seen in Figure 6.2. The AB provides domain B unrestricted dissemination rights, thereby allowing domain B the ability to further extend the trust relationship. More specifically, as the security level $High$ and Low of the AB trusts the security level $b1$ and $b3$ of domain B, respectively, it forms cross-lattice relationships $High \geq b1$ and $Low \geq b3$ and issues the following name certificates to map $High$ to $b1$ and Low to $b3$:

$$K_{MLS\_AB}\ High \longrightarrow K_{MLS\_B}\ b1$$

$$K_{MLS\_AB}\ Low \longrightarrow K_{MLS\_B}\ b3$$

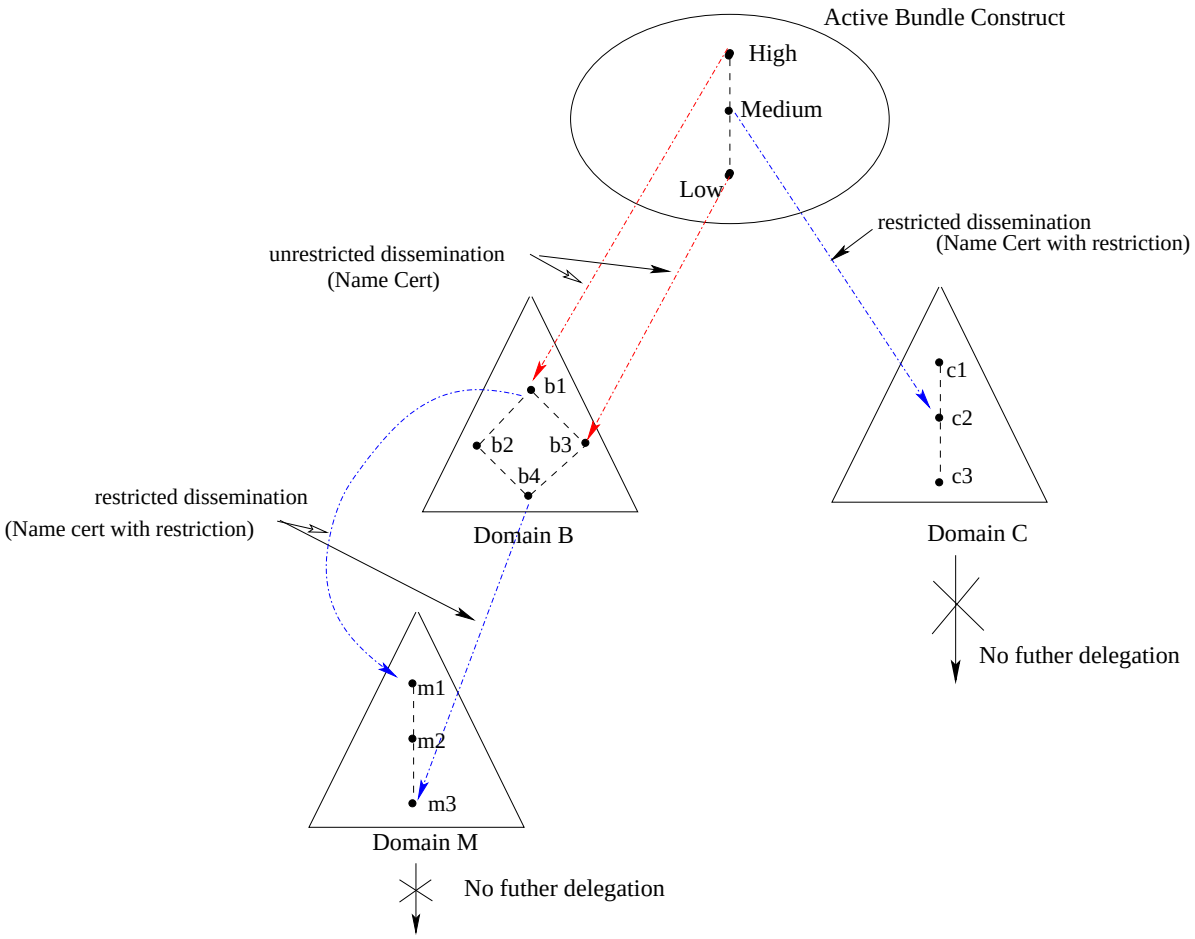


Figure 6.2: MLS Trust relationships (mappings)

Domain B has to decide whether it will delegate the authorization rights, from domain A, to domain M (unrestricted dissemination) or not (restricted dissemination). In the scenario shown in Figure 6.2, domain B decides to issue restricted dissemination to domain M. Thus, domain M cannot delegate the trust further to another domains. To indicate the restricted dissemination, a simple synthetic sugar (modification) to the SPKI name certificate is required. For example, domain B issues the following name certificates

to map $b1$ to $m1$ and $b4$ to $m3$:

$$K_{MLS\_B}\ b1 \longrightarrow K_{MLS\_M}\ m1,\ K_{MLS\_M}\ m1 \longrightarrow K_{M\_m1} \tag{6.1}$$

$$K_{MLS\_B}\ b4 \longrightarrow K_{MLS\_M}\ m3,\ K_{MLS\_M}\ m3 \longrightarrow K_{M\_m3} \tag{6.2}$$

As shown in the above equations (6.1) and (6.2), the synthetic sugar is added to the name certificates issued and signed by $K_{MLS\_B}\ b1$ and $K_{MLS\_B}\ b4$. For example, the restriction $K_{MLS\_M}\ m1 \longrightarrow K_{M\_m1}$ requires that the security level $m1$ of domain M must be assigned to an individual entity (public key) which belongs to the security group $K_{MLS\_M}\ m1$. This restriction has the following implications:

- $K_{MLS\_M}\ m1$ cannot be mapped to the security levels of other domains and must be reduced to a public key or entity eligible for the security level $K_{MLS\_M}\ m1$
- The restricted name certificate signals to the domains wanting to access to an object in the AB that it should contact other trusted domains to build a trust (or cross-lattice) relationship.
- To shorten the restricted name certificate, the superscript $*$ is added at the end of he mapped security level–eliminate the extra name (reduction). For example, the above equations (6.1) and (6.2) can be shortened as below.

$$K_{MLS\_B}\ b1 \longrightarrow K_{MLS\_M}\ m1^{*} \tag{6.3}$$

$$K_{MLS\_B}\ b4 \longrightarrow K_{MLS\_M}\ m3^{*} \tag{6.4}$$

  Note that the security level labeled with $*$ always shows up last in the name certificate chain.

MLS Data Dissemination Policy Specification

Definition 1. *The security level mapping G_{SMap} is a directed acyclic graph (V,E) where the vertex $v \in V$ represents a security level of a domain and the edge $e = (l_i, l_j)$ denotes a dominance relationship $l_i \geq l_j$ or a security level mapping $l_i \longrightarrow l_j$ between two different security levels. Note that l_i and l_j are the security levels of two different domains, namely, $l_i \in L_i$ and $l_j \in L_j$. A path in G_{SMap} from vertex l_1 to vertex l_n is a sequence of edges such that the start vertex of the first edge is $l_1 \in L_1$, the end vertex of the last edge is $l_n \in L_n$ and the start vertex of the $(i+1)^{st}$ edge is the same as the end vertex of the i^{th} edge. This path is denoted as: $p(l_1, l_n) = l_1 \mapsto l_2 \mapsto \cdots \mapsto l_n$.*

To specify the MLS data dissemination policy, the following predicates are needed:

- $path(l_1, l_n, D_x) \equiv true$ if there exists a path $p(l_1, l_n)$ in G_{SMap}. This means that G_{SMap} has the following path: $l_1 \mapsto \cdots \mapsto l_n$, where $l_1 \in L_{AB}$ and $l_n \in L_x$. Note that the path always starts at the active bundle domain, $l_1 \in L_{AB}$.
- $certificateChain(l_1, l_n, D_x) \equiv \exists l_1 \in L_{AB}, l_n \in L_x.path(l_1, l_n, D_x)$
- $dominate(l_1, l_2, D_x) \equiv true$ if $l_1 \geq_x l_2$, where $l_1, l_2 \in L_x$.
- $mlsAuth_{AB}(l_i, l_j, op) \equiv true$ if $(dominate(l_i, l_j, D_{AB}) \wedge op = read)$ or $(dominate(l_j, l_i, D_{AB}) \wedge op = write)$.
- To grant access to the object in D_{AB}, there must exist a security label l of D_{AB} which eventually maps to the security level $sl_x(s)$ of the subject s in D_x and the same security level l must satisfy the dominance requirement of D_{AB}:

$$accessGrant_{AB}(s, D_x, ob, op) \equiv$$
$$\exists l \in L_{AB}.(certificateChain(l, sl_x(s)) \wedge mlsAuth_{AB}(l, sl_{AB}(ob), op)),$$
$$where\ ob \in Resource_{AB}$$

Example 12. *As shown in Figure 6.2, there are cross-lattice relationships from the AB to domain M. For example, Low* $\geq b3$ *and* $b4 \geq m3$*. With the internal lattice relationship* $b3 \geq b4$*, we have Low* $\geq m3$*. The following name certificates are issued by the AB, domain B:*

$$K_{MLS\_AB}\ Low \longrightarrow K_{MLS\_B}\ b3 \tag{6.5}$$

$$K_{MLS\_B}\ b3 \longrightarrow K_{MLS\_B}\ b4 \tag{6.6}$$

$$K_{MLS\_B}\ b4 \longrightarrow K_{MLS\_M}\ m3* \tag{6.7}$$

Note that the name certificate (6.6) is issued by domain B to represent the inter-lattice relationship and publicly available. The third name certificate (6.7) for restricted dissemination requires that domain M, more specifically, $K_{MLS\_M}\ m3$ *is not allowed for further delegation and is able to issue the name certificate to an individual entity (public key) which belongs to the security group* $K_{MLS\_M}\ m3$*.*

Assume that Mike's security clearance level is K_{MLS_m3} *and his public key is* $K_{M\_m3\_Mike}$ *and he wants to access an object* K_{MLS_AB} *Low* o_2 *in the AB. Then he should provide the above three name certificates and* $K_{MLS\_M}\ m3 \longrightarrow K_{M\_m3\_Mike}$*. Then this forms the following certificate chain, which can satisfy the* $certificateChain$ *predicate:*

$$K_{MLS\_AB}\ Low \longrightarrow K_{MLS\_B}\ b3 \longrightarrow K_{MLS\_B}\ b4 \longrightarrow K_{MLS\_M}\ m3*$$

With the last name certificate, $K_{MLS\_M}\ m3 \longrightarrow K_{M\_m3\_Mike}$*, we have:*

$$K_{MLS\_AB}\ Low \longrightarrow K_{MLS\_B}\ b3 \longrightarrow K_{MLS\_B}\ b4 \longrightarrow K_{MLS\_M}\ m3* \longrightarrow K_{M\_m3\_Mike}$$

Finally, there exists a security level $K_{MLS\_AB}\ Low$ *in the above chain that dominates*

the security level K_{MLS_AB} Low of the object, it satisfies the $mlsAuth_{AB}$ predicates. Thus the access is granted ($accessGrant_{AB}$ predicate becomes true) by the active bundle.

MLS Data Dissemination Example

Unrestricted MLS Data Dissemination Example

In the following example, the AB trusts the Army and anyone the Army trusts; therefore, the Army has unrestricted dissemination rights. Meaning the Army can issue certificates to other domains, thus extending the trust.

The AB stores both its authorization policy (SPKI authorization certificates) and trust policy (SPKI name certificates) within its metadata. For this example, the AB (K_{MLS_AB}) has two data objects (O_1 *and* O_2), each with its own security label (*High and Low*) respectively. The authorization policy below states: *High* can read O_1 and *Low* can read O_2 respectively.

$$K_{MLS\_AB}\ \square \xrightarrow[\text{read } O_1]{} K_{MLS\_AB}\ High\ \blacksquare \tag{6.1}$$

$$K_{MLS\_AB}\ \square \xrightarrow[\text{read } O_2]{} K_{MLS\_AB}\ Low\ \blacksquare \tag{6.2}$$

The trust policy below states: AB *High* trusts Army $TopSecret : \{bio\}$ and AB *Low* trusts Army $Secret : \{\}$ respectively.

$$K_{MLS\_AB}\ High \longrightarrow K_{MLS\ ARMY}\ TopSecret : \{bio\} \tag{6.1}$$

$$K_{MLS\_AB}\ Low \longrightarrow K_{MLS\ ARMY}\ Secret : \{\} \tag{6.2}$$

Example 13. *The AB creator determines the data objects contained within the AB along with assigning the classification (security label) of each data object (Figure 6.3 step 1). The AB creator then develops the authorization policy, establishing the privileges to each data object (Figure 6.3 step 2). Next, the AB creator develops the trust policy, determining the entities the AB will directly trust (Figure 6.3 step 3). Finally, all certificates (authorization certificates and name certificates) are written to the directory server (Figure 6.3 step 4).*

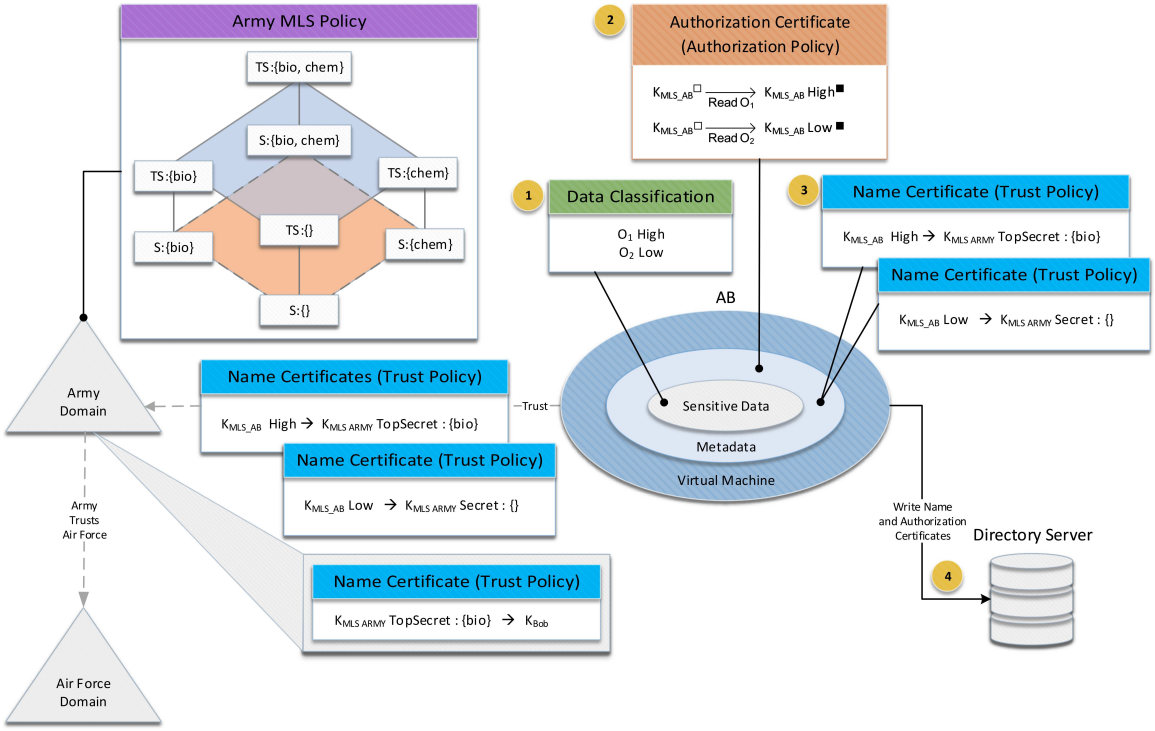


Figure 6.3: Example of creating an AB containing MLS data and policies.

In this example, Alice (K_{Alice}) from the Air Force wants to access O_1 within the AB (K_{MLS_AB}). The Air Force does not have a direct trust relationship with the AB. Therefore, the Air Force can learn about the authorization and trust policies directly from the AB or by sending a query to the directory server.

Example 14. *The AB arrives at the Air Force domain and is enabled by the Air Force (Figure 6.4 step 1). At this point, the Air Force can learn about the AB directly through the AB or from the directory server (Figure 6.4 step 2). The Air Force determines the Army has the appropriate level of trust and authorization required to access O_1 of the AB. Furthermore, the Air Force downloads a copy of the name certificate issued by the AB to the Army (this certificate will be required as proof when making the request for access in step 6). The Air Force issues a request to the Army for a name certificate showing trust between the Army and the Air Force (Figure 6.4 step 3). The Army trusts the Air Force; therefore, a name certificate is issued to the Air Force (Figure 6.4 step 4) and sent to the directory server (Figure 6.4 step 5). Alice makes the request to the AB and provides the certificate chain that proves trust (Figure 6.4 step 6). The AB verifies the name certificate chain, in order to validate the trust (Figure 6.4 step 7). Once trust is verified the AB then consults its Authorization policy to determine if Alice is authorized to read O_1 (Figure 6.4 step 8). Finally, the AB allows Alice to read O_1 (Figure 6.4 step 9).*

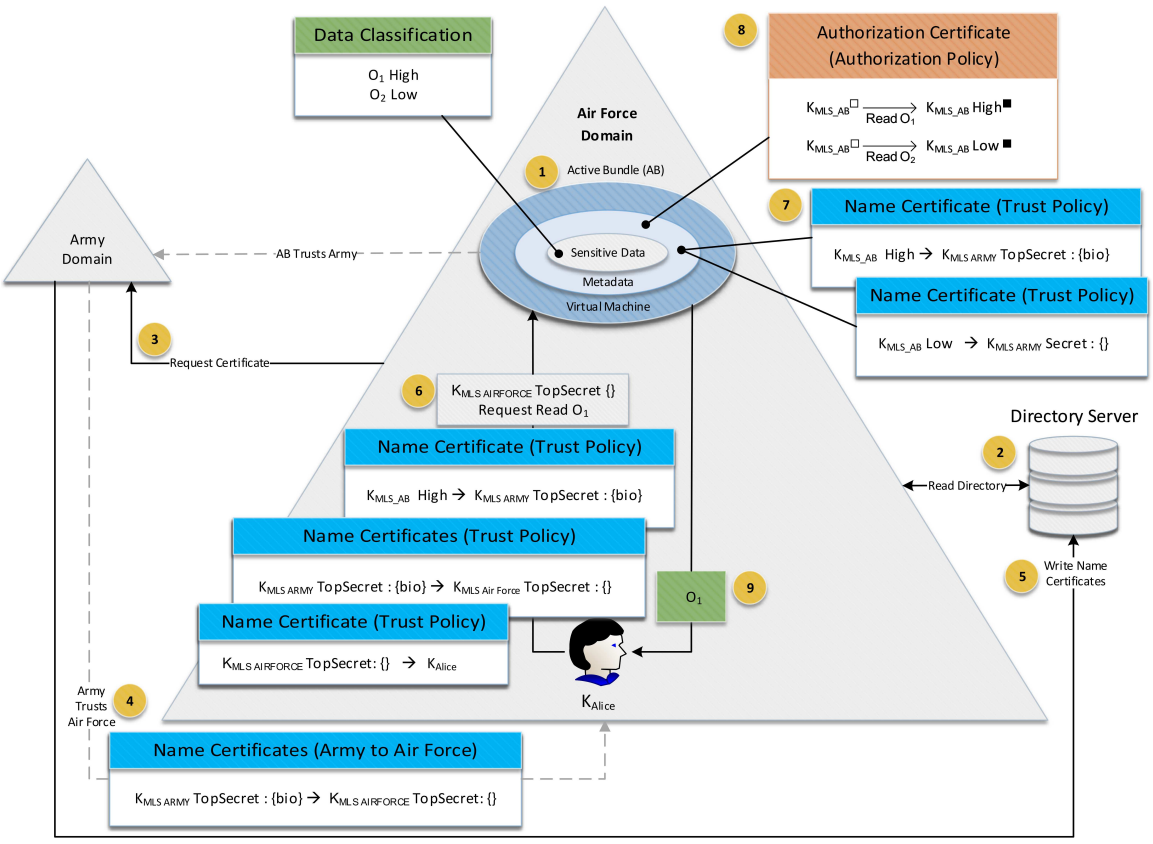


Figure 6.4: Example of AB activation with unrestricted dissemination.

Restricted MLS Data Dissemination Example

In the following example, the AB trusts the Army, however, the AB wants to limit the trust to only the Army. Meaning the Army can only issue certificates within its domain to its own subjects. The Army must not issue certificates to subjects outside the Army's domain.

Once again, the AB (K_{MLS_AB}) has two data objects (O_1 *and* O_2), each with its own security label (*High and Low*) respectively. The authorization policy below states: *High* can read O_1 and *Low* can read O_2 respectively.

$$K_{MLS\_AB}\ \square \xrightarrow[\text{read } O_1]{} K_{MLS\_AB}\ High\ \blacksquare \tag{6.1}$$

$$K_{MLS\_AB}\ \square \xrightarrow[\text{read } O_2]{} K_{MLS\_AB}\ Low\ \blacksquare \tag{6.2}$$

The following trust policy includes two certificates. The first certificate below states: AB *High* trusts Army $TopSecret : \{bio\}$, however, the AB wants to limit the trust to only the Army. Essentially line two below is stating that $K_{MLS\ ARMY}\ TopSecret : \{bio\}$ can only be mapped to subjects within the Army domain. If a certificate chain is sent to the AB and signed by another domain other than the Army it will be invalid.

$$K_{MLS\_AB}\ High \longrightarrow K_{MLS\ ARMY}\ TopSecret : \{bio\} \tag{6.1}$$

$$K_{MLS\ ARMY}\ TopSecret : \{bio\} \longrightarrow K_{ARMY TopSecret:\{bio\}} \tag{6.2}$$

The second certificate states AB *Low* trusts Army $Secret : \{\}$, and again the AB

wants to limit the trust to only the Army.

$$K_{MLS\_AB}\ Low \longrightarrow K_{MLS\ ARMY}\ Secret : \{\} \tag{6.1}$$

$$K_{MLS\ ARMY}\ Secret : \{\} \longrightarrow K_{ARMYSecret:\{\}} \tag{6.2}$$

Example 15. *The AB creation process is similar to the process for unrestricted dissemination, except the trust policy contains the restriction highlighted in red (Figure 6.5 step 3).*

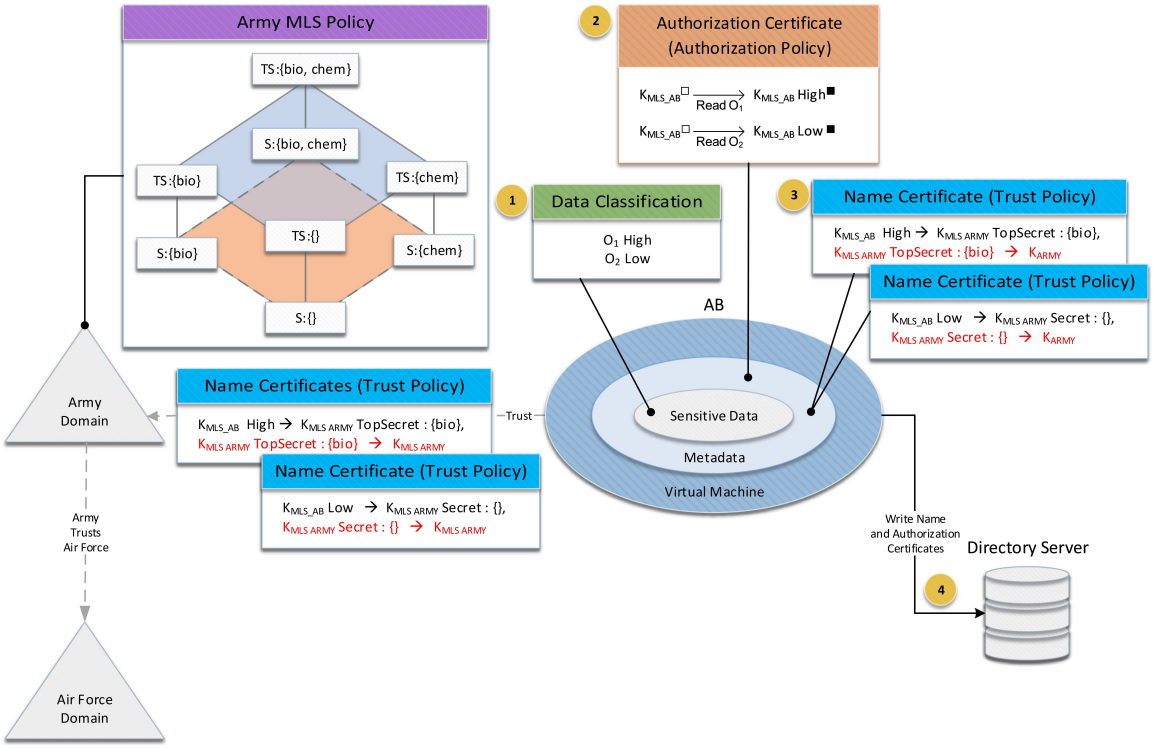


Figure 6.5: Example of creating an AB containing MLS data and policies.

In this example, Bob from the Army (K_{Bob}) wants to access O_1 within the AB (K_{MLS_AB}). Bob is a subject within the Army domain and has been issued a certificate from the Army domain, allowing Bob access to $TopSecret : \{bio\}$.

Example 16. *The AB arrives at the Army domain and is enabled by the Army (Figure 6.6 step 1). At this point, Bob (K_{Bob}) can learn about the AB directly through the AB or from the directory server (Figure 6.6 step 2). Bob makes the request to the AB to read O_1 and provides the certificate chain that proves trust (Figure 6.6 step 3). The AB verifies the name certificate chain, in order to validate the trust (Figure 6.6 step 4). Once trust is verified the AB then consults its authorization policy to determine if Bob is authorized to read O_1 (Figure 6.6 step 5). Finally, the AB allows Bob to read O_1 (Figure 6.6 step 6).*

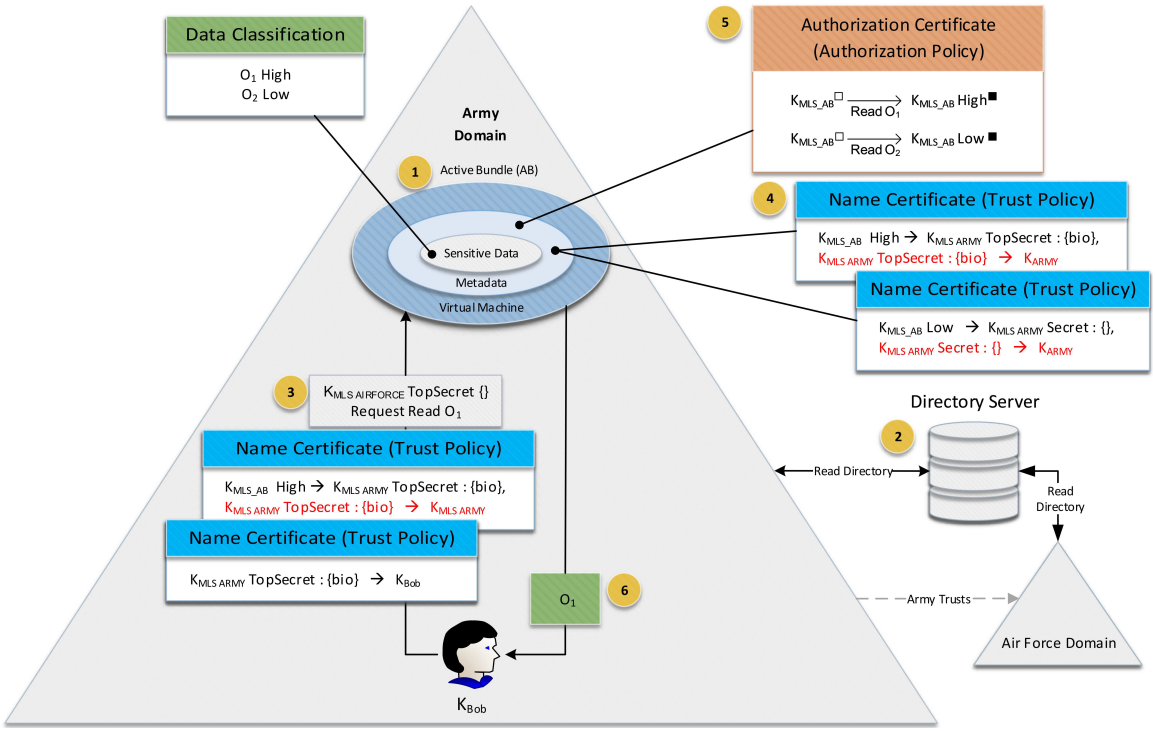


Figure 6.6: Example of AB activation with restricted dissemination.

In Example 16, the Army user Bob was able to read O_1 as Bob certificate chain was verified. In the below certificate chain (Figure 6.7) it's clear to see that Bob is trusted by $K_{MLS\ ARMY}\ TopSecret : \{bio\}$, Furthermore, the second certificate would be signed by the Army domain, thus meeting the restriction from certificate 1. However, in Figure 6.8 Alice is issued a certificate from the Air Force domain, thus it is signed by the Air Force and does not satisfy the restriction placed in the first certificate.

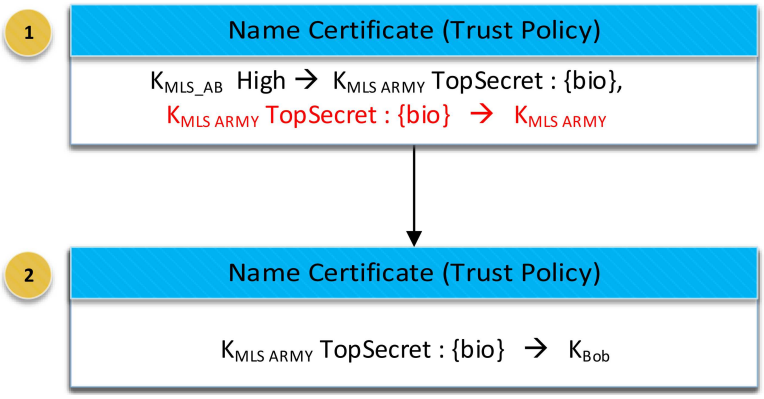


Figure 6.7: Valid certificate chain.

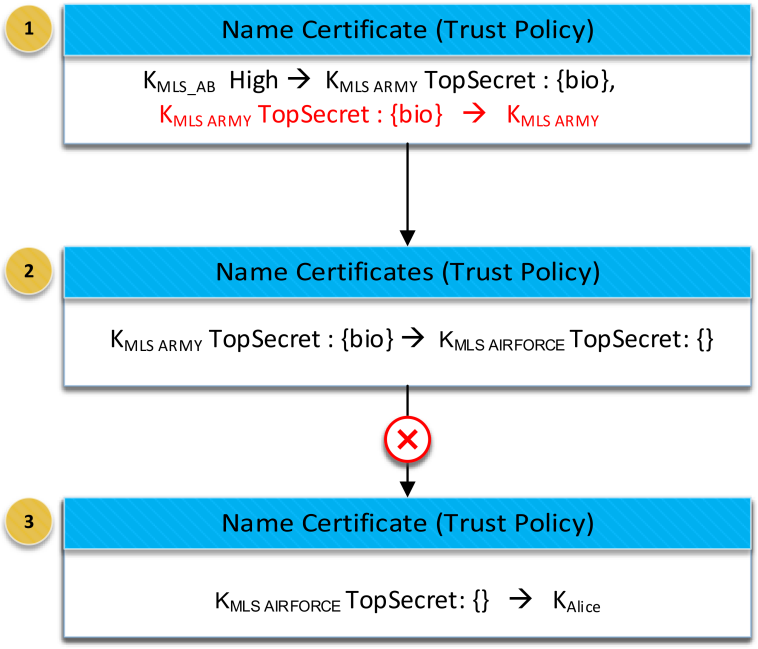


Figure 6.8: Invalid certificate chain.

Each domain is trusted to issues certificates to subjects within their own domains only. Cross-domain certificates should only be issued between domains in order to demonstrate a trust relationship between the two domains. Therefore, domain *A* should issue certificates to subjects of domain *A* but not to subjects of domain *B*. Domain *A* should issue a certificate to domain *B* showing the trust between the two domains. However, if domain *A* issues a certificate to a subject of domain *B* there is no mechanism in place to detect or prevent this malicious activity. Domains must trust one another and not issue certificates to subjects outside their domain.

Summary

This project set out to identify a potential solution for MLS data dissemination by leveraging existing technologies and concepts. However, prior to addressing a solution, six test scenarios were developed that identified various violations that can occur from incorrect mapping. The six test scenarios demonstrate the impacts on security related

to incorrect mapping, specifically, how the direction of trust and the privileges play a role in violations. The six test scenarios contribute by providing additional insight into violations that was missing within the literature review. Although the test scenarios provided valuable insight, more research is needed to fully understand and formalize the violations. Additionally, the MLS policy are represented using SPKI name and authorization certificates and tested using Prolog. Leveraging Prolog allowed for testing the logic behind SPKI certificates, and contributed to the overall solution by providing a logical model. Additional prototyping is necessary to ensure the mechanics work in a real-world setting. However, as mentioned previously a small modification to the SPKI name certificate is necessary to accommodate restricted dissemination. Finally, bringing together MLS, SPKI, Active Bundle, and a directory server provided a proposed solution, additional research is required to refine the theory. For example, throughout this project, the directory server is mentioned several times. However, the mechanics behind how it will provide the functions necessary for domains to exchange MLS related information was not covered and more research is required.

APPENDIX A

ABBREVIATIONS

AB	Active Bundle
AES	Advanced Encryption Standard
BLP	Bell-La Padula
DAC	Discretionary Access Control
MAC	Mandatory Access Control
MLS	Multi-Level Security
PRNG	Pseudo Random Number Generator
SDSI	Simple Distributed Security Infrastructure
SPKI	Simple Public Key Infrastructure
RBAC	Role-Based Access Control
VM	Virtual Machine

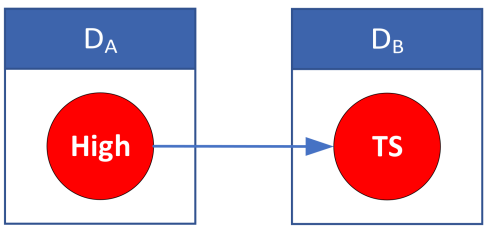
APPENDIX B

TEST SCENARIO NOTATION

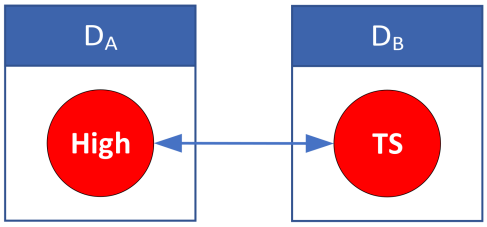
Symbol	Description
	Solid line represents full trust (read and write).
	Short-dashed line represents partial trust allowing read.
	Long-dashed line represents partial trust allowing write.
	The arrow depicts a one-way trust in the direction of arrow. It points from the trustor (the entity that is trusting) to the trustee (the entity that is being trusted).
	The arrow depicts a two-way trust.
	Dotted line with an arrow represents either information flow (when bold) or read/write by pointing from the source to destination.
	Box represents the boundaries of the Domain.
	Circle represents a particulate subject/object with a security level. The color is used to highlight the correct mapping. Therefore, if both circles are red they have the same level

Notation examples The following examples depict two domains (D_A and D_B) each with a single security level ($High$ and TS). A domain with only one security level would be uncommon within an MLS environment; however, the examples are simplified in order to explain the notation.

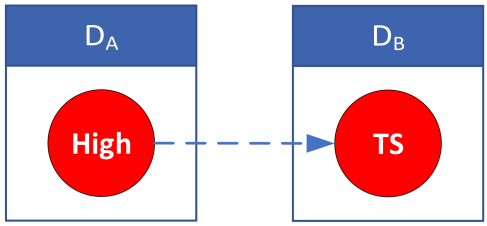
One-way full trust Domain D_A fully trusts domain D_B, therefore, D_A $High$ fully trusts D_B TS. The solid line depicts full trust, meaning both read and write privileges. The arrow depicts the direction of trust, in this case D_A $High$ is trusting D_B TS. Therefore, D_B TS can both read and write to D_A $High$, whereas D_A $High$ cannot read or write to D_B TS.



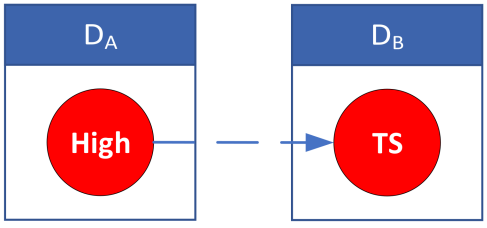
Two-way full trust Domain D_A fully trusts domain D_B, therefore, D_A *High* fully trusts D_B *TS*. In addition, domain D_B fully trusts domain D_A, therefore, D_B *TS* fully trusts D_A *High*. The solid line depicts full trust, meaning both read and write privileges. The arrow shows that D_A *High* is trusting D_B *TS*, and D_B *TS* is trusting D_A *High*. Therefore, D_B *TS* can both read and write to D_A *High*, and D_A *High* can read and write to D_B *TS*.



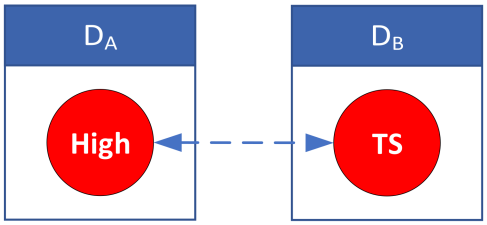
One-way partial trust (read) Domain D_A partially trusts domain D_B with read privileges, therefore, D_A *High* partially trusts D_B *TS*. The short-dashed line depicts partial trust in the form of read privileges, meaning D_B *TS* can read from D_A *High*, whereas D_A *High* cannot read or write to D_B *TS*.



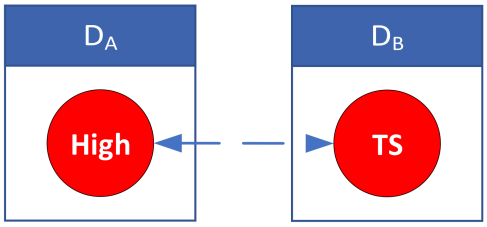
One-way partial trust (write) Domain D_A partially trusts domain D_B with write privileges, therefore, D_A *High* partially trusts D_B *TS*. The long-dashed line depicts partial trust in the form of write privileges, meaning D_B *TS* can write to D_A *High*, whereas D_A *High* cannot read or write to D_B *TS*.



Two-way partial trust (read) Domain D_A partially trusts domain D_B, therefore, D_A *High* partially trusts D_B *TS*. In addition, domain D_B partially trusts domain D_A, therefore, D_B *TS* partially trusts D_A *High*. The short-dashed line depicts partial trust in the form of read privileges, meaning D_B *TS* can read from D_A *High*, and D_A *High* can read from D_B *TS*.



Two-way partial trust (write) Domain D_A partially trusts domain D_B, therefore, D_A *High* partially trusts D_B *TS*. In addition, domain D_B partially trusts domain D_A, therefore, D_B *TS* partially trusts D_A *High*. The long-dashed line depicts partial trust in the form of write privileges, meaning D_B *TS* can write to D_A *High*, and D_A *High* can write to D_B *TS*.



APPENDIX C

PROLOG IMPLEMENTATION

Objects and Subjects

/*Active Bundle labels for Objects*/ (1)

object_sl(”K_AB_o1”,”K_AB_Low”). (2)

object_sl(”K_AB_o2”,”K_AB_Medium”). (3)

object_sl(”K_AB_o3”, ”K_AB_High”). (4)

/*Army Subject (Key, Clearance, Issuer)*/ (5)

subject_sl(”K_Alice”,”K_ARMY_TopSecret:{bio}”,”K_ARMY”). (6)

subject_sl(”K_Bob”,”K_ARMY_Secret:{bio}” ,”K_ARMY”). (7)

subject_sl(”K_Tom”,”K_ARMY_Secret:{}” ,”K_ARMY”). (8)

/*Air Force Subjects*/ (9)

subject_sl(”K_Kevin”,”K_AirForce_TopSecret:{}” ,”K_AirForce”). (10)

subject_sl(”K_Rob”,”K_AirForce_Secret:{}” ,”K_AirForce”). (11)

/*NAVY Subjects*/ (12)

subject_sl(”K_Nick”,”K_Navy_TopSecret:{}” ,”K_Navy”). (13)

subject_sl(”K_Nancy”,”K_Navy_Secret:{}” ,”K_Navy”). (14)

/*Marines*/ (15)

subject_sl(”K_Matt”,”K_Marien_TopSecret:{}” ,”K_Marine”). (16)

subject_sl(”K_Mark”,”K_Marine_Secret:{}” ,”K_Marine”). (17)

Authorization Policies (SPKI Auth Certs)

/*Authorization Policy auth_policy(Principal, Operation, Object, Subject)*/ (18)

/*High*/ (19)

auth_policy(”K_AB”,read,”K_AB_o1”,”K_AB_High”). (20)

auth_policy(”K_AB”,read,”K_AB_o2”,”K_AB_High”). (21)

auth_policy(”K_AB”,read,”K_AB_o3”,”K_AB_High”). (22)

auth_policy(”K_AB”,write,”K_AB_o3”,”K_AB_High”). (23)

/*Medium*/ (24)

auth_policy(”K_AB”,read,”K_AB_o1”,”K_AB_Medium”). (25)

auth_policy(”K_AB”,read,”K_AB_o2”,”K_AB_Medium”). (26)

auth_policy(”K_AB”,write,”K_AB_o2”,”K_AB_Medium”). (27)

auth_policy(”K_AB”,write,”K_AB_o3”,”K_AB_Medium”). (28)

/*Low*/ (29)

auth_policy(”K_AB”,read,”K_AB_o1”,”K_AB_Low”). (30)

auth_policy(”K_AB”,write,”K_AB_o1”,”K_AB_Low”). (31)

auth_policy(”K_AB”,write,”K_AB_o2”,”K_AB_Low”). (32)

auth_policy(”K_AB”,write,”K_AB_o3”,”K_AB_Low”). (33)

/*Restricted Dissemination*/ (34)

trust(”K_AB_Medium”, ”K_ARMY_Secret:{bio}”, (35)

trust(”K_ARMY_Secret:{bio}”,”K_ARMY”)). (36)

/*Unrestricted Dissemination*/ (37)

/*AB Trust Policy (Trustor, Trustee, Dissemination Restriction)*/ (38)

trust(”K_AB_High”, ”K_ARMY_TopSecret:{bio}”). (39)

trust(”K_AB_Low”, ”K_ARMY_Secret:{}”). (40)

/*Army Issued Trust Certificate to Air Force*/ (41)

trust(”K_ARMY_TopSecret:{bio}”,”K_AirForce_TopSecret:{}”). (42)

trust(”K_ARMY_Secret:{bio}”,”K_AirForce_Secret:{}”). (43)

/*Air Force Issued Trust Certificate to Navy*/ (44)

trust(”K_AirForce_TopSecret:{}”,”K_Navy_TopSecret:{}”). (45)

/*Navy Issued Trust Certificate to Mariens*/ (46)

trust(”K_Navy_TopSecret:{}”,”K_Marien_TopSecret:{}”). (47)

trusts(X, Y):- trust(X,Y, trust(Y,Z)), subject_sl(_,Y, Z),nl, (48)

write(’Restricted Dissemination’). (49)

trusts(X, Y):- trust(X,Y),nl, write(’Unrestricted Dissemination’). (50)

trusts(X, Y):- trust(X,Z), trust(Z,Y),nl, write(’Unrestricted Dissemination’). (51)

trusts(X, Y):- trust(X,Z), trusts(Z,Y),nl, write(’Unrestricted Dissemination’). (52)

```
/*If a trust relationship can be established*/                    (53)
accessGrant(Subject_Key, Operation, Object_Name) :-               (54)
    subject_sl(Subject_Key, Subject_sl,_),                        (55)
    object_sl(Object_Name, _),                                    (56)
    trusts(X, Subject_sl),                                        (57)
    auth_policy("K_AB",Operation,Object_Name, X).                 (58)
```

```
list_Auth_Policy:- auth_policy(Issuer, Operation, Object, Subject),                    (59)
    write('Issuer: '),tab(13),write(Issuer),nl,                                        (60)
    write('Operation: '),tab(7),write(Operation),nl,                                   (61)
    write('Object: '),tab(12),write(Object),nl,                                        (62)
    write('Subject: '),tab(10),write(Subject),nl.                                      (63)
list_Subjects:- subject_sl(Name, Clearance, Issuer),                                   (64)
    write('Subjet Key: '),tab(4),write(Name),nl,                                       (65)
    write('Clearance: '),tab(6),write(Clearance),nl,                                   (66)
    write('Issuer: '),tab(12),write(Issuer),nl.                                        (67)
list_Objects:- object_sl(Classification, Key),                                         (68)
    write("Object's Key: "),tab(6),write(Key),nl,                                      (69)
    write('Classification: '),tab(6),write(Classification),nl.                         (70)
list_Trust:- trust(Issuer, Subject), write('Issuer: '),tab(12),write(Issuer),nl,       (71)
    write('Subject: '),tab(10),write(Subject),nl,                                      (72)
    write('Unrestricted Dissemination'),nl.                                            (73)
list_Trust:- trust(Issuer, Subject,trust(_,Domain_Key)),                               (74)
    write('Issuer: '),tab(12),write(Issuer),nl,                                        (75)
    write('Subject: '),tab(10),write(Subject),nl,                                      (76)
    write('Valid Only if Issued By: '), write(Domain_Key),nl,                          (77)
    write('Restricted Dissemination'),nl.                                              (78)
```

APPENDIX D

NOTATION AND SYNTAX

A brief description of the notation and syntax used is provided within this appendix.

D_i denotes a domain whose name is i. For example, D_{Army}, and D_{AB} represent the domains of $Army$ and AB(Active Bundle) respectively. Note that we treat an Active Bundle construct as a domain.

$Resource_i$ represents a set of all the resources (both objects and subjects) in the domain D_i. Where i represents the domain. For example, R_{Army} denotes a set of all the objects and subjects of the domain D_{Army}.

O_i represents a single object within some domain, where i represents the name of the object. When referencing an object in this manner a reference to the domain will be included. For example, O_1 within the AB (K_{MLS_AB}) denotes O_1 is an object within the domain K_{MLS_AB}.

K_{MLS_Name} $SecLevel : \{ca\_set\}$ represents a security level or label of the domain D_{Name}.

- K_{MLS_Name} is a public key responsible for assigning security levels to resources in R_{Name}.
- $SecLevel$ denote the sensitivity (clearance/classification) level.
- ca_set denote the a set of categories or compartments of the domain D_{Name}. Note that ca_set is optional and can remain blank, $\{\}$.
- For example, K_{MLS_Army} $TopSecret : \{chem, nuke\}$ is a security level of the domain D_{Army}, with a sensitivity of Top Secret, and compartments of both chem and nuke. Furthermore, K_{MLS_Navy} $Confidential : \{bio\}$ is a security level of the domain D_{Navy} and has a sensitivity of Confidential with a compartment of bio.

L_i represents a single security level within a domain. For example l_i could represent $TS : \{bio, chem\}$ within the D_{Army}.

l_i represents the set of all the security levels or labels of domain D_i.

$\mathscr{L}_i$ represents a security level lattice of the domain D_i, $\mathscr{L}_i = \langle L_i, \geq_i \rangle$.

sl_i is a function which returns the security level of an object or subject in the domain D_i. For instance, $sl_{Army}(ob) = K_{Army}\ \ TS : \{bio, chem\}$, if $ob \in Resource_{Army}$.

$\geq_x$ represents a dominance relationship within an individual lattice $\mathscr{L}_x$ of the domain D_x. For example, $l_m \geq_x l_n$ means l_m, $l_n \in L_x$ and l_m dominates l_n.

BIBLIOGRAPHY

Denning, D. E. (1976). A lattice model of secure information flow. *Communications of the ACM*, *19*(5), 236–243.

Ellison, C. M., Frantz, B., Lampson, B., Rivest, R., Thomas, B. M., & Ylonen, T. (1999). Spki certificate theory. In *Internet rfc 2693*.

Dawson, S., Qian, S., & Samarati, P. (2000). Providing security and interoperation of heterogeneous systems. *Distributed and Parallel Databases*, *8*(1), 119–145. doi:10.1023/A:1008787317852

Crampton, J. (2002). *Authorization and antichains* (Doctoral dissertation, Birkbeck College, University of London).

Bell, D. E. (2005). Looking back at the bell-la padula model. In *Computer security applications conference, 21st annual* (15–pp). IEEE.

Shehab, M., Bertino, E., & Ghafoor, A. (2005). Serat: Secure role mapping technique for decentralized secure interoperability. In *Proceedings of the tenth acm symposium on access control models and technologies* (pp. 159–167). SACMAT '05. doi:10.1145/1063979.1064007

Hermanns, H., & Palsberg, J. (2006). *Tools and algorithms for the construction and analysis of systems: 12th international conference, tacas 2006, held as part of the joint european conferences on theory and practice of software, etaps 2006, vienna, austria, march 25 - april 2, 2006: Proceedings*. Springer.

Shafiq, B. (2006). Access control management and security in multi-domain collaborative environments. *ETD Collection for Purdue University*.

Kim, K. I., Kim, W. Y., Ryu, J. S., & Kim, U. M. (2009). Secure collaboration based on rbac in decentralized multi-domain environments. In *2009 1st ieee symposium on web society* (pp. 200–204). doi:10.1109/SWS.2009.5271798

Othman, L. B., & Lilien, L. (2009). Protecting privacy in sensitive data dissemination with active bundles. In *Proc. 7th annual conf. on privacy , trust and management of e-business* (pp. 202–213).

Othman, L. B. (2010). Active bundles for protecting confidentiality of sensitive data throughout their lifecycle. In *Ph.d dissertation. department of computer science, western michigan university, kalamazoo, mi*.

Salih, R. M., Lilien, L., & Othman, L. B. (2012). Protecting patients' electronic health records using enhanced active bundles. In *Proceedings of the 6th international conference on pervasive computing technologies for healthcare(pervasivehealth)*.

Wooden, C. (2012). Pope hosts top-level meeting on leaks in vatican. Retrieved from https://www.catholicnews.com/services/englishnews/2012/pope-hosts-top-level-meeting-on-leaks-in-vatican.cfm

Gouglidis, A., Mavridis, I., & Hu, V. C. (2013). Verification of secure inter-operation properties in multi-domain rbac systems. In *2013 ieee seventh international conference on software security and reliability companion* (pp. 35–44). doi:10.1109/SERE-C.2013.25

Liguori, A. (2016). *Design and implementation of multilevel security architectures* (Doctoral dissertation, UNIVERSITY OF ROMA TRE).

Hilia, M., Chibani, A., Winter, T., & Djouani, K. (2017). Semantic based authorization framework for multi-domain collaborative cloud environments. *Procedia Computer Science*, *109*, 718–724. 8th International Conference on Ambient Systems, Networks and Technologies, ANT-2017 and the 7th International Conference

on Sustainable Energy Information Technology, SEIT 2017, 16-19 May 2017, Madeira, Portugal. doi:https://doi.org/10.1016/j.procs.2017.05.427

Sarhan, A. Y., & Carr, S. (2017). A highly-secure self-protection data scheme in clouds using active data bundles and agent-based secure multi-party computation. In *4th international conference on cyber security and cloud computing (cscloud)* (pp. 228–236). IEEE.

Sanger, D. E., Perlroth, N., Thrush, G., & Rappeport, A. (2018). Marriott data breach is traced to chinese hackers as u.s. readies crackdown on beijing, The New York Times. Retrieved from https://www.nytimes.com/2018/12/11/us/politics/trump-china-trade.html

2016 Presidential Campaign Hacking Fast Facts. (2019). Cable News Network. Retrieved from https://www.cnn.com/2016/12/26/us/2016-presidential-campaign-hacking-fast-facts/index.html